FPGA IMPLEMENTATION OF LDPC CODES

ABHISHEK KUMAR

211EC2081

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

Rourkela-769008,

Odisha, INDIA

FPGA IMPLEMENTATION OF LDPC CODES

A dissertation submitted in partial fulfilment of the
requirement for the degree of

"Master of Technology"
in
VLSI Design and Embedded System

Submitted by

ABHISHEK KUMAR
211EC2081

Under the Guidance of
Dr. SARAT KUMAR PATRA

Department of Electronics and Communication Engineering

National Institute of Technology, Rourkela

Rourkela-769008, Odisha, INDIA

DEDICATED TO *MY LOVING PARENTS AND MY SISTER*

DECLARATION

I certify that

1. The work contained in this thesis is original and has been done by me under the guidance of my supervisor (s).
2. The work has not been submitted to any other Institute for the award of any other degree or diploma.
3. I have followed the guidelines provided by the Institute I preparing the thesis.
4. I have confirmed to the norms and guidelines in the Ethical Code of Conduct of the Institute.
5. Whenever I used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

ABHISHEK KUMAR
211EC2081
Rourkela, June'13

Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela

C E R T I F I C A T E

This is to certify that the thesis entitled "FPGA Implementation of LDPC Codes" being submitted by Mr. ABHISHEK KUMAR, to the National Institute of Technology, Rourkela (Deemed University) for the award of degree of Master of Technology in Electronics and Communication Engineering with specialization in "VLSI Design and Embedded System", is a bonafide research work carried out by him in the Department of Electronics and Communication Engineering, under my supervision and guidance. I believe that this thesis fulfils a part of the requirements for the award of degree of Master of Technology. The research reports and the results embodied in this thesis have not been submitted in parts or full to any other University or Institute for the award of any other degree or diploma.

_____

Dr. Sarat Kumar Patra
Dept. of Electronics & Communication.
National Institute of Technology
Rourkela, Odisha, 76900

Place: N.I.T., Rourkela
Date:

ACKNOWLEDGEMENTS

First and foremost, I am truly indebted and wish to express my gratitude to my supervisor Professor Sarat Kumar Patra for his inspiration, excellent guidance, continuing encouragement and unwavering confidence and support during every stage of this endeavour without which, it would not have been possible for me to complete this undertaking successfully. I also thank him for his insightful comments and suggestions which continually helped me to improve my understanding.

I express my deep gratitude to the members of Masters Scrutiny Committee, Professors D. P. Acharya, and A. K. Swain for their loving advice and support. I am very much obliged to the Head of the Department of Electronics and Communication Engineering, NIT Rourkela for providing all possible facilities towards this work. Thanks to all other faculty members in the department.

I would like to express my heartfelt gratitude to Madhusmita Mishra who kept me in focus and helped a lot in the project on several occasions. I would also like to express my heartfelt gratitude to my friend and senior Soumya Ranjan Biswal who have inspired me and particularly helped in the project.

My wholehearted gratitude to my parents, my sister and my friends for their constant love, encouragement, and support.

Above all, I thank Almighty who bestowed his blessings upon us.

ABHISHEK KUMAR
211EC2081
Rourkela, June’13

ABSTRACT

Low density parity check (LDPC) codes are linear block codes used for error detection and correction mostly in high speed digital communication systems like digital broadcasting, optical fibre communications and wireless local area networks. LDPC codes have been subject to extensive research because of their significant performance in error correction.

LDPC Code is a type of Block Error Correction code discovered and performance very close to Shanon's limit .Good error correcting performance enables reliable communication. Since its discovery by Gallagar there is more research going on for its efficient construction and implementation. Though there is no unique method for constructing LDPC codes. Implementation of LDPC Code is done by taking different factors in to consideration such as error rate, parallelism of decoder, ease in implementation etc.

This thesis is about FPGA implementation of LDPC codes and their performance evaluation. Protograph codes were introduced and analyzed by NASA's Jet Propulsion Laboratory in the early years of this century. Part of this thesis continues that work, investigating the decoding of specific protograph codes and extending existing tools for analyzing codes to protograph codes

In this thesis I have taken the performance of LDPC coded BPSK modulated signal which is transmitted through AWGN channel and the performance is tested using MATLAB Simulation

Table of Contents

Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1. Historical background

In his seminal 1948 paper, Claude Shannon derived the mathematical laws that govern how rapidly information can be reliably transmitted through a noisy channel. This mathematical framework became the basis for an entirely new field called information theory, devoted to its study and its sister discipline error-correcting codes.

Shannon's noisy channel coding theorem asserts that for every channel there exists a maximum rate at which we can communicate with vanishing error probabilities. This maximum rate is known as the capacity of the channel. Shannon further proved that this capacity can be achieved by almost any extremely long code. This proof, however, was not constructive. An arbitrary long, random code may technically perform well, but the encoding and decoding times would be prohibitively large.

In the decades following Shannon's work, the ultimate goal of coding theory has been to construct capacity-achieving codes with manageable encoding and decoding times. One major success in this endeavour was the introduction of turbo codes in 1993. With turbo codes, came the introduction of iterative decoding, which bridged the gap between high performance and low complexity. Specifically, iterative decoding can achieve performance close to theoretical limits with a complexity that grows only linearly with the length of the code.

The discovery of turbo codes led to a flurry of research interest in the field, and, in particular, to the rediscovery of Gallager's 1963 work on low-density parity check (LDPC) codes. Though Gallager's work had been largely forgotten due to the limited computational capabilities of his time,

some interesting developments had been occurring. Most relevant to this thesis was the work of Tanner, which formally introduced the idea of using a bipartite graph to graphically represent a code.

The idea of irregular codes was introduced in 1998 by Luby et al. as a way to improve upon Gallager's regular codes. Five years later, NASA's Jet Propulsion Lab (JPL) introduced the idea of a protograph code. A protograph code is more structured than an irregular code, which allows for simpler code descriptions without sacrificing performance. Protograph codes are closely related to Tanner's codes created from seed graphs, and are an example of the multi-edge type construction introduced by Richardson.

With new codes came new theorems explaining their success. LDPC codes, with iterative decoding, have been shown to achieve excellent performance over many channels, nearly approaching capacity on additive white Gaussian noise (AWGN) channel, and as code's length tends to infinity, achieving it on the binary erasure channel (BEC).

1.2. Scope of This Thesis

In this chapter, we will provide the necessary background information that the rest of the thesis depends on.

Communication system transmits data from source to transmitter through a channel or medium such as wired or wireless. The reliability of received data depends on the channel medium and external noise and this noise creates interference to the signal and introduces errors in transmitted data. Shannon through his coding theorem showed that reliable transmission could be achieved only if data rate is less than that of channel capacity. The theorem shows that a sequence of codes of rate less than the channel capacity have the capability as the code length goes to infinity. Error detection and correction can be achieved by adding redundant symbols to the original data called as error correction and correction codes (ECCs).Without ECCs data need to retransmitted if it could detect there is an error in the received data. ECC are also called as for error correction (FEC) as we can correct bits without retransmission. Retransmission adds delay, cost and wastes system throughput. ECCs are really helpful for the long distance one way communications such as deep space communications or satellite communications. They also have application in wireless communication and storage devices.

1.3. Error Detection and Correction Schemes

Error detection and correction helps in transmitting data in a noisy channel to transmit data without errors. Error detection refers to detect errors if any received by the receiver and correction is to correct errors received by the receiver.

Different errors correcting codes are there and can be used depending on the properties of the system and the application in which the error correcting is to be introduced. Generally error correcting codes have been classified into block codes and convolutional codes. The distinguishing feature for the classification is the presence or absence of memory in the encoders for the two codes.

To generate a block code, the incoming information stream is divided into blocks and each block is processed individually by adding redundancy in accordance with a prescribed algorithm. The decoder processes each block individually and corrects errors by exploiting redundancy.

In a convolutional code, the encoding operation may be viewed as the discrete–time convolution of the input sequence with the impulse response of the encoder. The duration of the impulse response equals the memory of the encoder. Accordingly, the encoder for a convolutional code operates on the incoming message sequence, using a sliding window equal in duration to its own memory. Hence in a convolutional code, unlike a block code where code words are produced on a block-by-block basis, the channel encoder accepts message bits as continuous sequence and thereby generates a continuous sequence of encoded bits at a higher rate.

An error-correcting code (ECC) or forward error correction (FEC) code is a system of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors (up to the capability of the code being used) were introduced, either during the process of transmission, or on storage. Since the receiver does not have to ask the sender for retransmission of the data, a back-channel is not required in forward error correction, and it is therefore suitable for simplex communication such as broadcasting. Error-correcting codes are frequently used in lower-layer communication, as well as for reliable storage in media such as CDs,
DVDs, hard disks, and RAM.

1.4. Linear Block Codes

Linear block coding is a subtype of block coding that is made by dividing the information sequence into message blocks. Linear block codes have a linear algebraic structure that provides a reduction in the encoding and decoding complexity compared to arbitrary block codes.

Definition 1.

An (n, k) linear block code ζ with message word length k and codeword length n over the finite field F2 = ({0, 1}, +, ·) is a k dimensional subspace of the vector space V (F2), of n-tuples with elements from F2. There are 2k message words u = [u0n, u1, …, uk-1] and 2k corresponding code words c = [c0,c1, …, cn-1] in the code ζ. Thus a linear code of length n is a subspace of Vn which is spanned by k linearly independent vectors g0, g1, …, gk-1 of Vn. With the k linearly independent vectors g, …, gk-1 of V given above, any codeword X can be written as a linear combination of these vectors as follows...

$$X = \sum_{i=1}^{k} migi \quad (1)$$

Different code words are obtained for different combinations of the coefficients of m. Also the codeword X can be represented by matrix multiplication as X=mG where m is a 1 by k matrix (vector) which is essentially the message word to be encoded and G is a k by n matrix whose rows constitute the k linearly independent vectors gi's. G is called the generating matrix of ζ. From the above discussion, it is easy to see that G has rank k, hence it can be reduced to the form G = [Ik | P] where Ik is a k by k identity matrix. The reduction of G to that form may need some column swapping which permutes the order of the bits in the code words.

In addition, using G matrix, if a message word m is encoded to a codeword ζ, then the first k bits of ζ are exactly equal to m. This results an easy extraction of original message sent after decoding a received word. The null space ζ~ of the subspace ζ has dimension n-k and is spanned by n-k linearly independent vectors h0, h1,…, hn-k-1. Since each hi belongs to ζ~, for any c in ζ, $c.h_i^T = 0$ for all i. Furthermore, if x is any binary block of length n but x does not belong to ζ, then $x.h_i^T \neq 0$ for all i. These n-k linearly independent vectors hi, constitute the rows of a matrix called Parity Check Matrix so that $c.h_i^T = 0$, if and only if c belongs to ζ.

Definition 2.

The syndrome of a codeword x is defined as the product of x with the transpose of the parity check matrix H like, $S = x \cdot HT = 0$. Thus upon arrival, a received word is valid if and only if its syndrome is zero. A generating matrix G in the form of G =[I | A] so that the first k bits of any codeword x are exactly equal to the message word it encodes and the parity check matrix is H = [AkT | I]. Syndrome decoding is used in LDPC decoding algorithms when deciding if the decoded codeword is correct or not.

1.5. Low Density Parity Check Codes

LDPC codes are linear block codes specified by a sparse parity check matrix. This means the number of 1's per column (column weight) is very small compared to the column length of parity check matrix and the number of 1's per row (row weight) is very small compared to the row length of parity check matrix.

LDPC codes are classified into two groups like regular LDPC codes and irregular LPDC codes according to the row and column weight properties of parity check matrix. In regular LDPC codes, the parity check matrix has uniform column weight and row weight. On the contrary, in irregular LDPC codes the parity check matrix has non-uniform column weight and row weight. As the result of extensive research done on regular and irregular LDPC codes, it is found that irregular LDPC codes have a better error correcting performance than regular LDPC codes. On the other hand, regular LPDC codes have the advantage of regularity which brings them a big advantage like they can be implemented much easier compared to irregular LDPC codes. LDPC decoder implementations presented in this thesis have irregular LDPC(quasi-cyclic) code structure.

Besides the parity check matrix representation, LDPC codes can be represented by a bipartite graph called Tanner graph. A bipartite graph is a graph whose nodes may be separated into two classes, and where edges may only be connecting two nodes not residing in the same class. The two classes of nodes in a Tanner graph are bit nodes and check nodes. The Tanner graph of a code is drawn according to the following rule: Check node fj ; j = 1,...,.N - K is connected to bit node xi; i = 1,...,N whenever element h in H (parity check matrix) is a one. Edges of the Tanner graph act as information path between bit nodes and check nodes for decoding process. Figure 1 shows a Tanner graph made

for a simple parity check matrix H. In this graph each bit node is connected to two check nodes and each check node is connected to four bit nodes.

LDPC codes are constructed by defining the parity check matrix H. If the parity check matrix A has N columns and M rows, any codeword generated for this LDPC code consists of N bits which satisfy M parity checks, where the location of a 1 in the parity check matrix indicates that a bit is involved in a parity check. The total length of the codeword is N bits, the number of message bits is K = N - M, and the rate of the code is R = K / N, assuming that the matrix is full rank.

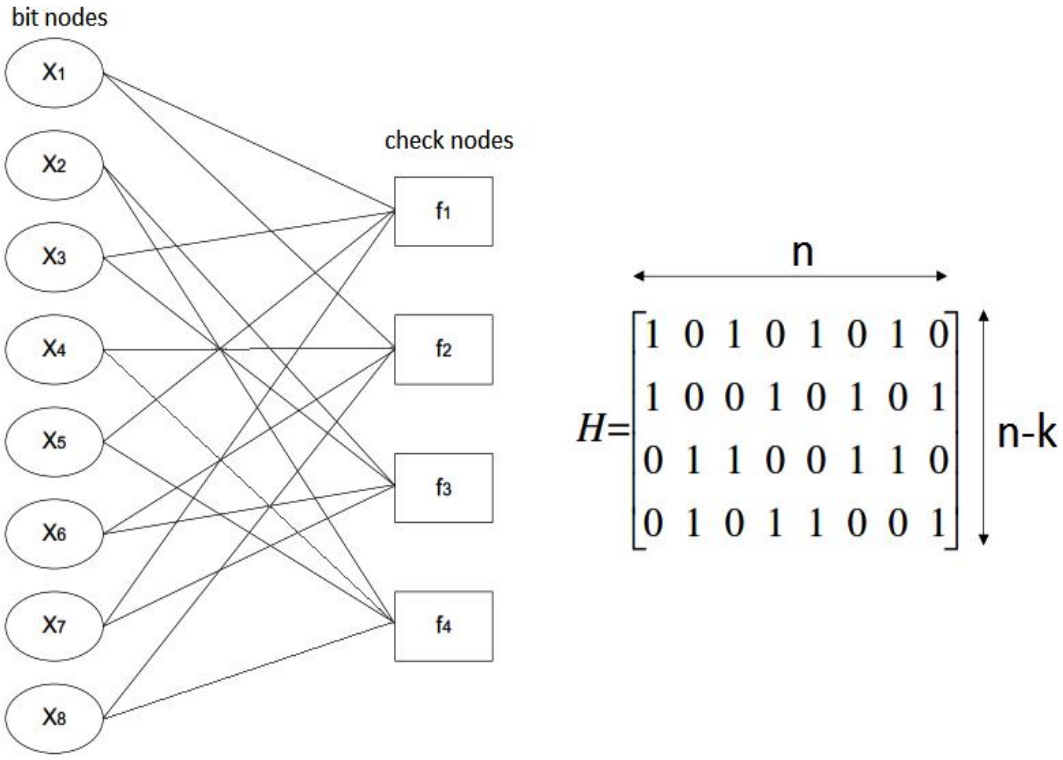


Figure 1 Tanner graph made for a simple parity check matrix H

1.6. Protograph codes

Many protographs which look structurally different have equivalent spectral shapes. Figure 2. shows three protographs representing ensembles which are contained within the regular (3,6) ensemble. The difference between the three protograph ensembles is that the ensemble featured on the left has no codes which contain double edges, while the centre and right ensembles do contain double edges. However, all three of these code ensembles share their spectral shape with that of regular (3,6) ensemble. While a regular ensemble always contains more codes than a protograph representation of the same ensemble, the difference is slight and cannot be distinguished in spectral shape.

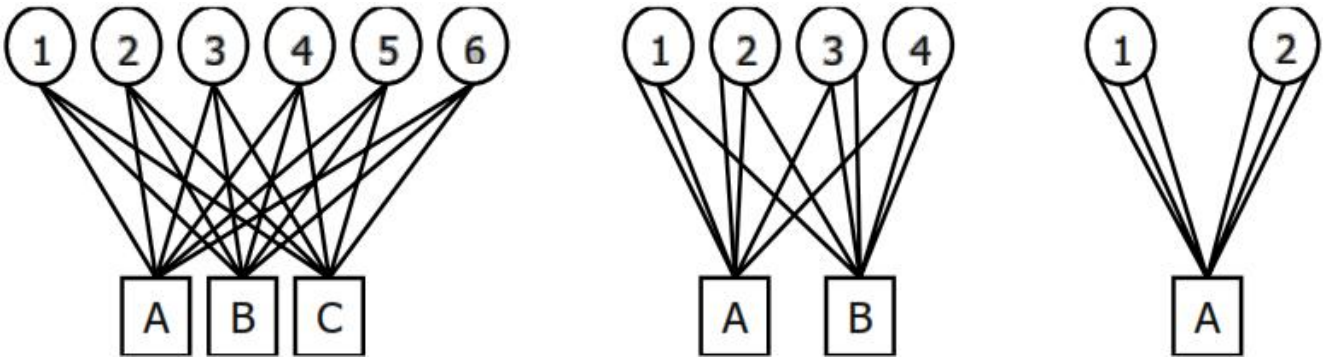


Figure 2 shows three protographs representing ensembles contained within the regular (3,6) ensembles.

1.6.1. AR4JA protograph

The AR4JA LDPC codes proposed in this document posses relatively large minimum distance for their block length and undetected error rates lie several orders of magnitude below detected frame and bit error rates for any given operating signal-to noise ratio.

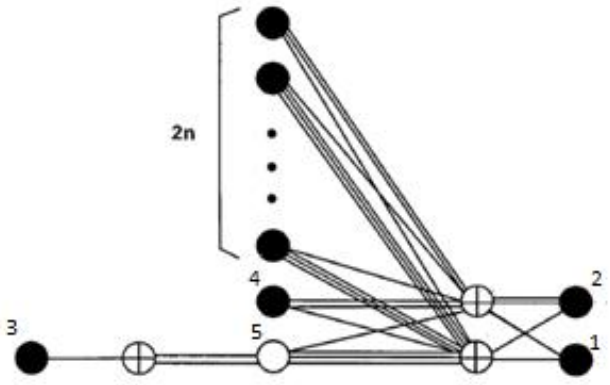


Figure 3 Protograph of AR4JA code family

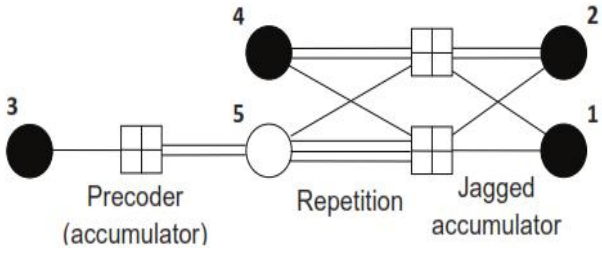


Figure 4 Protograph for AR4JA for rate 1/2

1.6.2. Expanding and realising the protograph

A direct QC expansion of the AR4JA protograph shown in matrix below will create a QC LDPC code. The AR4JA codes defined in the experimental CCSDS standard use a two step expansion process. After a first cyclic expansion by a factor of 4, a new larger type-I weight matrix obtained as shown in matrix 1 for rate-½.

The first 4 rows correspond to check node number 1 in figure 4, the second 4 rows and the last 4 rows correspond to check nodes 2 and 3, respectively. The first 4 columns correspond to variable

node number 1 in figure 4. The subsequent 4 groups of 4 columns correspond to variable node numbers 2, 3, 4 and 5 respectively in figure 4.

A *type-I* weight matrix is one that contains only ones and zeros meaning that the associated protograph has no parallel edges. According to the CCSDS standard, the matrix 1 is expanded in a second step cyclic expansion to create the three block lengths, corresponding to k=1024 information bits, QC LDPC code. In this final expansion, the scalar parity check matrix, H, is created by replacing each 1 entry of matrix 1 by a cyclic permutation submatrix

$$H=\begin{bmatrix} 0 & 0 & 2 & 3 & 1 \\ 1 & 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 1 & 3 \end{bmatrix}$$

$$\mathbf{A}=\left[\begin{array}{cccc|cccc|cccc|cccc|cccc} 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&1 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1 \\ \hline 1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1 \\ 0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&1&1 \\ 0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&0&1 \\ 0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&1&1&1&0 \\ \hline 1&0&0&0&0&0&1&1&0&0&0&0&1&1&0&0&1&0&0&0 \\ 0&1&0&0&1&0&0&1&0&0&0&0&0&1&1&0&0&1&0&0 \\ 0&0&1&0&1&1&0&0&0&0&0&0&0&0&1&1&0&0&1&0 \\ 0&0&0&1&0&1&1&0&0&0&0&0&1&0&0&1&0&0&0&1 \end{array}\right]$$

These codes are QC with a sub block size equal to the second step expansion factor (k). In other words, the two-step process is not equivalent to any single step cyclic expansion. Hence after the expansion according to the desired specifications the resultant matrix has a dimension of(3072 X 5120) and the matrix contains a total of 15230 non zero elements “1”.

1.7. Codes used in this thesis

The first LDPC code here is made from circulant matrices which are square matrices of binary entries, where each row is a one-position right cyclic shift of the previous row. Hence the entire circulant is determined by its first row, and low-weight circulants are used to define the parity check matrices with low density. The parity check matrix for rate 1/2 is shown in figure 5. below.

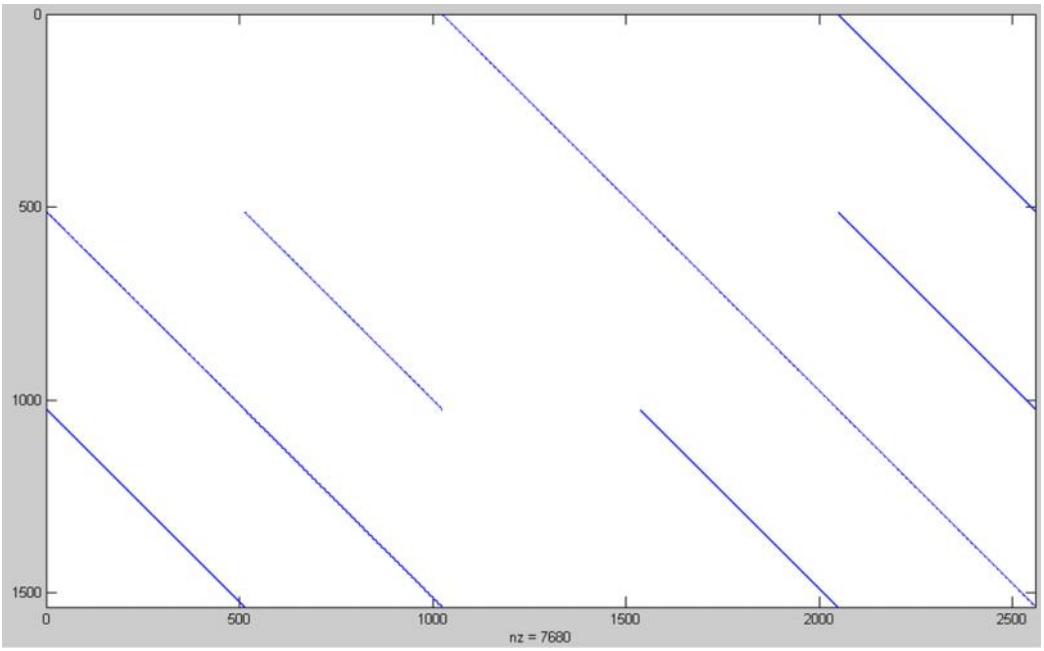


Figure 5 H matrix for code type 1

The second LDPC code here is of QC-LDPC type and it has cyclic properties in its sub-blocks fed irregularly. The sub-blocks are random in nature. The parity check matrix for rate 1/2 is shown in figure 6. below.

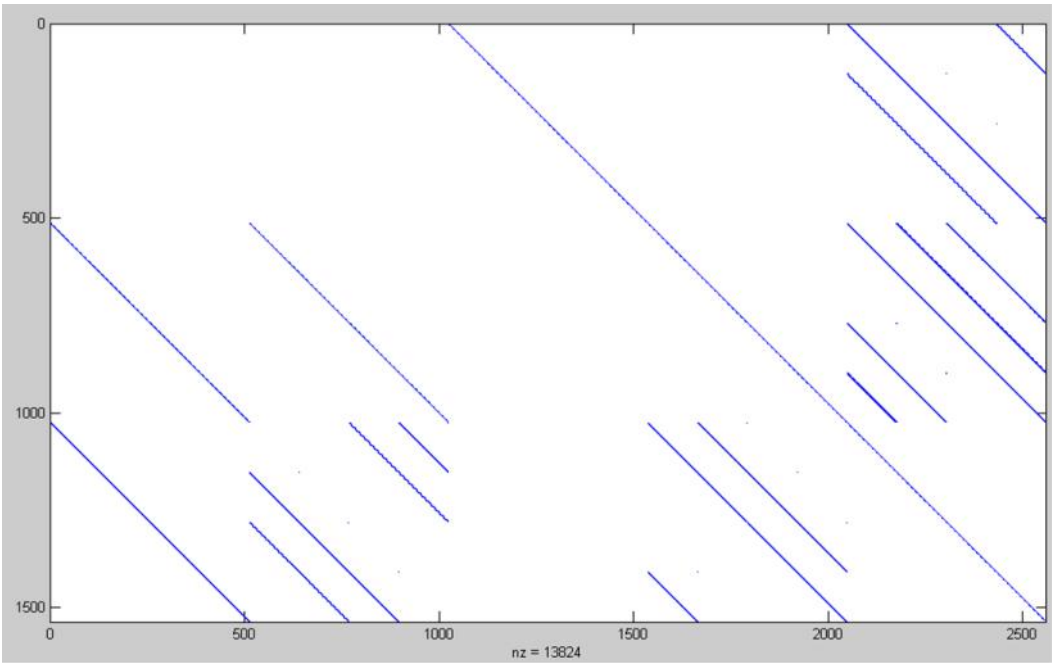


Figure 6 H matrix for code type 2

The third type AR4JA LDPC code combines the structure Quasi cyclic added with permutation basing on the basic protograph structure. The various code rates are generated by expanding, copying

and permuting the protograph structure. The parity check matrix for rate 1/2 is shown in figure 6 below. The parity check matrix of this code is similar in shape to that of second code with the difference that here the sub blocks are related through permutation to make a systematic structure. The advantage of this code over the second code is that the BER convergence is faster with lesser number of decoder iterations.

The *H* matrices for the rate-1/2 codes are specified as follows

$$H_{1/2} = \begin{bmatrix} 0_M & 0_M & I_M & 0_M & I_M \oplus \Pi_1 \\ I_M & I_M & 0_M & I_M & \Pi_2 \oplus \Pi_3 \oplus \Pi_4 \\ I_M & \Pi_5 \oplus \Pi_6 & 0_M & \Pi_7 \oplus \Pi_8 & I_M \end{bmatrix}$$

Where IM and 0M are identity and zero matrices respectively of size M. Π_1 to Π_8 are given by the equation 2:

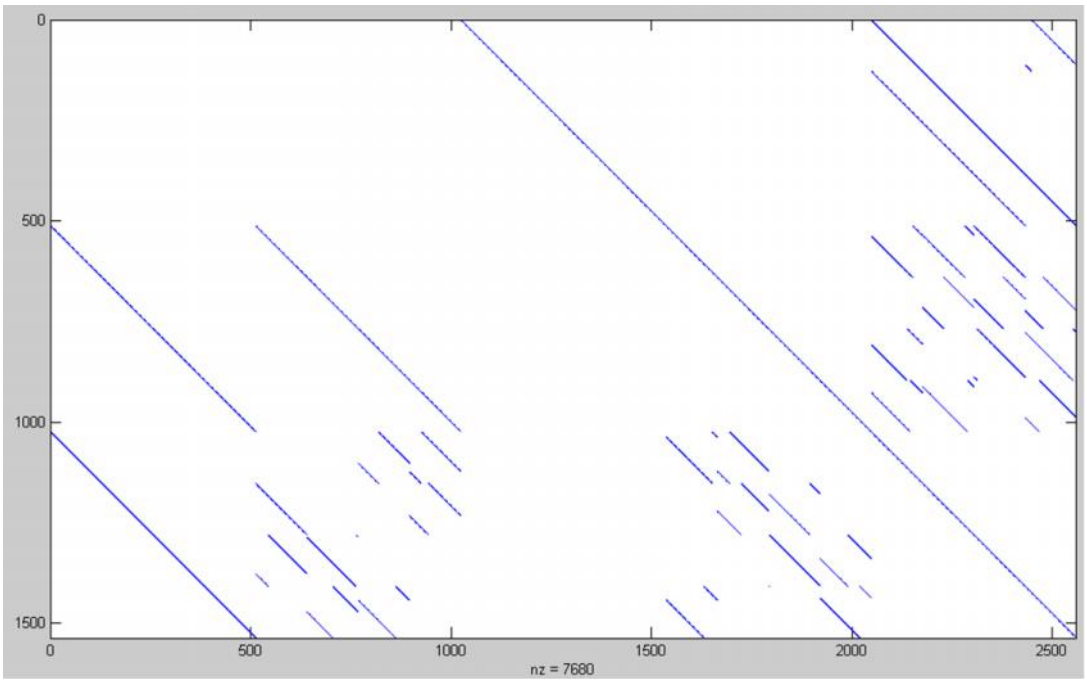


Figure 7 H matrix for code type 3(AR4JA)

$$\pi_k(i) = \frac{M}{4}\left[\left(\theta_k + \left\lfloor\frac{4i}{M}\right\rfloor\right) mod 4\right] + \left(\phi_k\left(\left\lfloor\frac{4i}{M}\right\rfloor, M\right) + i\right) mod \frac{M}{4} \quad \ldots (2)$$

Where, permutation matrix Πk has non zero entry in row i and column πk(i) for i = 0 to M-1. For different submatrix sizes M= {128,256,512,1024}, the values of θk and Φk are given in [3].

The H matrices for different rates are given as:

$$II_{2/3} = \left[\begin{array}{cc|c} 0_M & 0_M & \\ \Pi_9 \oplus \Pi_{10} \oplus \Pi_{11} & I_M & II_{1/2} \\ I_M & \Pi_{12} \oplus \Pi_{13} \oplus \Pi_{14} & \end{array}\right]$$

$$H_{3/4} = \left[\begin{array}{cc|c} 0_M & 0_M & \\ \Pi_{15} \oplus \Pi_{16} \oplus \Pi_{17} & I_M & H_{2/3} \\ I_M & \Pi_{18} \oplus \Pi_{19} \oplus \Pi_{20} & \end{array}\right]$$

$$H_{4/5} = \left[\begin{array}{cc|c} 0_M & 0_M & \\ \Pi_{21} \oplus \Pi_{22} \oplus \Pi_{23} & I_M & H_{3/4} \\ I_M & \Pi_{24} \oplus \Pi_{25} \oplus \Pi_{26} & \end{array}\right]$$

Table 1 Codeblock length (bits) for supported code rates

	Code block length(n)		
Information block length (k)	Rate ½	Rate 2/3	Rate 4/5
1024	512	1536	1280
2048	1024	3072	2560
4096	2048	6144	5120

Table 2 Values of Submatrix Size M for Supported Codes

	Submatrix size (M)		
Information block length (k)	Rate ½	Rate 2/3	Rate 4/5
1024	512	256	128
2048	1024	512	256
4096	2048	1024	512

The fourth type is the modified AR4JA code, where each of the non-empty sub matrices of AR4JA matrix are replaced by the same submatrix structure of one fourth size. The main difference between this modified AR4JA matrix and the AR4JA matrix is that here the sub matrix is quasi cyclic in nature while that in AR4JA matrix is circulant in nature. The advantage of this structure over

AR4JA is that here the BER performance is better than AR4JA in the low SNR region. The parity check matrix for rate 1/2 is shown in figure 8 below.

The permutation matrix equation $\pi_k(i)$ of this matrix type is given as:

$$\pi_k(i) = \frac{M}{4}\left[\left(\theta_k + \left\lfloor\frac{4i}{M}\right\rfloor\right) mod 4\right] + \frac{M'}{4}\left[\left(\theta_k + \left\lfloor\frac{4i}{M'}\right\rfloor\right) mod 4\right] + \left(\phi_k\left(\left\lfloor\frac{4i'}{M'}\right\rfloor, M'\right) + i'\right) mod \frac{M'}{4} \quad \text{... (3)}$$

Where, M'=M/4 and i'=0,1,....,M'-1while the operators and table remain the same.

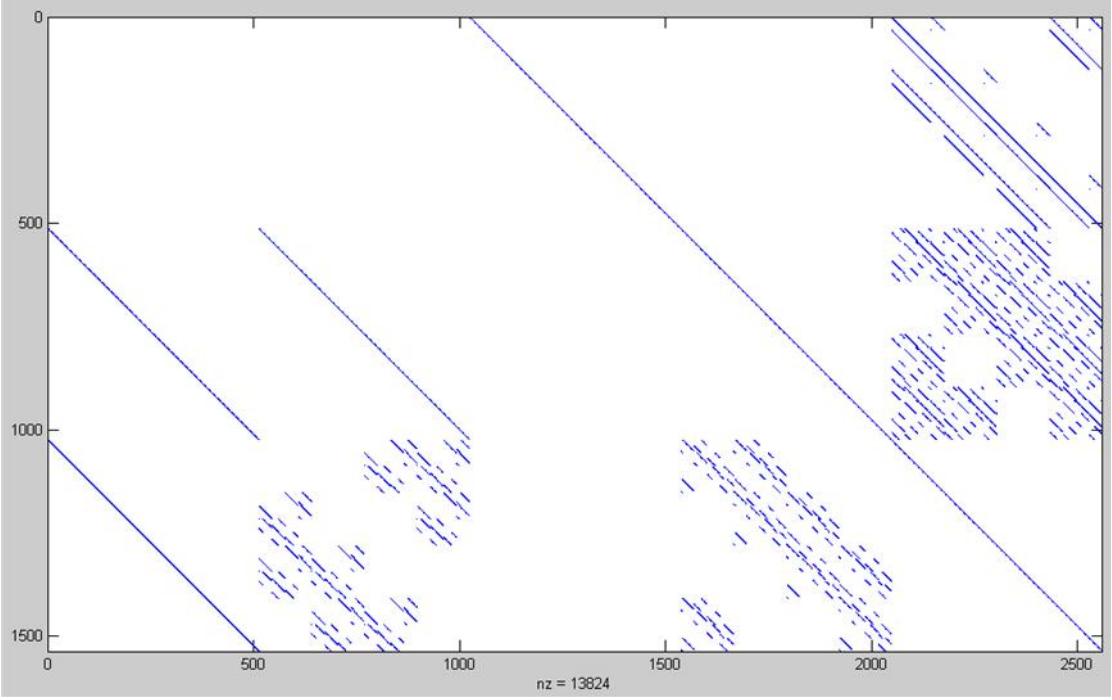


Figure 8 H matrix for code type 4(modified AR4JA)

CHAPTER 2

ENCODER

LDPC encoding is more complex than it appears for LDPC codes of big codeword lengths due to the computational intensity of matrix multiplication of generating matrix G and message word. There is extensive research done on low complexity encoding techniques based on the H matrix and efficient methods for LDPC encoding which can be found in the literature. Besides low complexity, it is also important that the encoding process should be suitable for different channels. Since the decoder implementations are made for AWGN channels in this thesis, encoding for AWGN channel is described briefly below.

Given a message word m, a corresponding codeword c such that $c = m \cdot G$ is generated. This codeword is then converted to integer numbers {-1, +1} word x according to the following rule: $x_i = (-1)^{c_i}$. This integer codeword is then sent through the channel and white Gaussian noise $n \sim N(0, s)$ is added to it. The resulting word has same length but the bits can have any real values that are caused due to the noise. Once decoding is done the codeword sent is recovered by inverse relation $c_{2i} = 0$ if $y_i = +1$ and $c_{2i} = 1$ if $y_i = -1$.

2.1. Circulant of H matrix

A circulant is a square matrix in which each row is the cyclic shift (one place to the right) of the row above it, and the first row is the cyclic shift of the last row. For such a circulant, each column is the downward cyclic shift of the column on its left, and the first column is the cyclic shift of the last column. The row and column weights of a circulant are the same, say w. For simplicity, we say that the circulant has weight .If , w=1 then the circulant is a permutation matrix, called a circulant permutation matrix. For a circulant, the set of columns (reading top-down) is the same as the set of

rows (reading from right to left). A circulant is completely characterized by its first row (or first column), which is called the *generator* of the circulant.

2.2. Codeword generation

For a b x b circulant over GF(2), if its rank is b, then all its rows are linearly independent. A QC-LDPC code is given by the null space of an array of sparse circulants of the same size. For two positive integers c and t with c ≤ t, consider the following c x t array of b x b circulants over GF(2):

$$\mathbf{H}_{\mathrm{qc}} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,t} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \cdots & \mathbf{A}_{2,t} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{c,1} & \mathbf{A}_{c,2} & \cdots & \mathbf{A}_{c,t} \end{bmatrix}$$

which has the following structural properties: 1) the weight of each circulant is small compared with its size ; and 2) no two rows (or two columns) of have more than one 1-component in common, called the row-column (RC) constraint.

2.3. Finding generator matrix

Consider the QC-LDPC code C_{qc} given by the null space of the parity-check matrix H_{qc} given by (1). Suppose the rank of is equal to cb . We assume that the columns of circulants of H_{qc} are arranged in such a way that the rank of the following sub array c x c of H_{qc} is cb , the same as the rank of H_{qc}. We also assume that the first (t-c)b columns of H_{qc} correspond to the (t-c)b information bits. The desired generator matrix of H_{qc} has the following form:

$$\mathbf{G}_{\mathrm{qc}} = \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \vdots \\ \mathbf{G}_{t-c} \end{bmatrix} = \left[\begin{array}{cccc|cccc} \mathbf{I} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{G}_{1,1} & \mathbf{G}_{1,2} & \cdots & \mathbf{G}_{1,c} \\ \mathbf{O} & \mathbf{I} & \cdots & \mathbf{O} & \mathbf{G}_{2,1} & \mathbf{G}_{2,2} & \cdots & \mathbf{G}_{2,c} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{I} & \mathbf{G}_{t-c,1} & \mathbf{G}_{t-c,2} & \cdots & \mathbf{G}_{t-c,c} \end{array}\right] = [\mathbf{I}_{(t-c)b} \quad \mathbf{P}]$$

Where I is a identity matrix, O is a zero matrix, and $G_{i,j}$ with $1 \leq i \leq t-c$ and $1 \leq j \leq c$ is a b x b circulant.

The necessary and sufficient condition for G_{qc} to be a generator matrix of C_{qc} is that $H_{qc}G_{qc}^T = [0]$, where [0] is a zero matrix.

Let $g_{i,j}$ be the generator of the circulant. Once we know $g_{i,j}$'s, we can form all the circulants $G_{i,j}$'s of G_{qc}. Therefore, G_{qc} is completely characterized by a set of c(t-c) circulant generators, which are called the generators of C_{qc}.

Let u = (1,0,...,0) be the unit b-tuple with a "1" at the first position, and 0=(0,...,0) be the all-zero b-tuple. For, $1 \leq i \leq t-c$ the first row of the submatrix of G_i is

$$g_i = (0 \dots 0\ u\ 0 \dots 0\ g_{i,1}\ g_{i,2} \dots\ g_{i,c})$$

where the unit b-tuple u is at the ith position of g_i.

The $H_{qc}\ g_i^T = 0$ gives the following equality:

$$M_i\ u^T + D\ z_i^T = 0$$

Which gives: $z_i^T = D^{-1}M_i u^T$ … (4).

Where, for $1 \leq i \leq t-c$, $z_i = (g_{i,1}\ g_{i,2} \dots\ g_{i,c})$ (ie. The last c sections of g_i) and the ith column of circulants of H_{qc} given by $M_i = \left[A_{1,i}^T \dots\ A_{c,i}^T\right]^T$.

Solving equation 4.we obtain z_1, z_2, z_{t-c}, for, $1 \leq i \leq t-c$. From z_1, z_2, z_{t-c}, we can find all $g_{i,j}$'s from which G_{qc} can be easily constructed.

2.4. Encoding process

The encoding process deals with the task to generate the systematic codeword for the input message string (a) applied to the input block. This process can be given by the following equation:

$$C = a\,G \quad \ldots (5).$$

In hardware implementation the input will be given one bit at a time. Which forces us to write the previous equation as:

$$C = a_i G_{i,j} = a_{(i-1)b+1} g_{i,j}^{(0)} + a_{(i-1)b+2} g_{i,j}^{(1)} + \cdots + a_{(ib)} g_{i,j}^{(b-1)} \quad \ldots (6).$$

Increasing demands of high speed communication systems and reduction of device sizes, increases stress on hardware developers to create more and more compact devices that does more computations on the same resources available.

Implementing the hardware for encoding requires a register array to accommodate entire generator matrix. Meeting the today's demands it is essential that more number of messages are passed which leads to larger generator matrix resulting larger memory use of hardware and hence it is unfavourable. To overcome this problem the H matrix is in use today is of cyclic in nature. The generator matrix will be cyclic then and hence it will be favourable to store just one row or column of that matrix. This row (preferred against column) is called generator of the circulant. The next row will be one bit right cyclic shift to this row and so on.

The hardware implementation figure is given as figure 9. The steps to encode a message string is given as:

1. On the positive going clock cycle edge the new row is loaded in the generator matrix register.
2. The input message bit is AND'ed with the contents of generator matrix register.
3. The contents of temporary output register are then XOR'ed with the previous step output.
4. The output in step 4 is then again stored in the same register.
5. This process is continued till all the rows of generator matrix are traversed once.

This process is given as the flow diagram

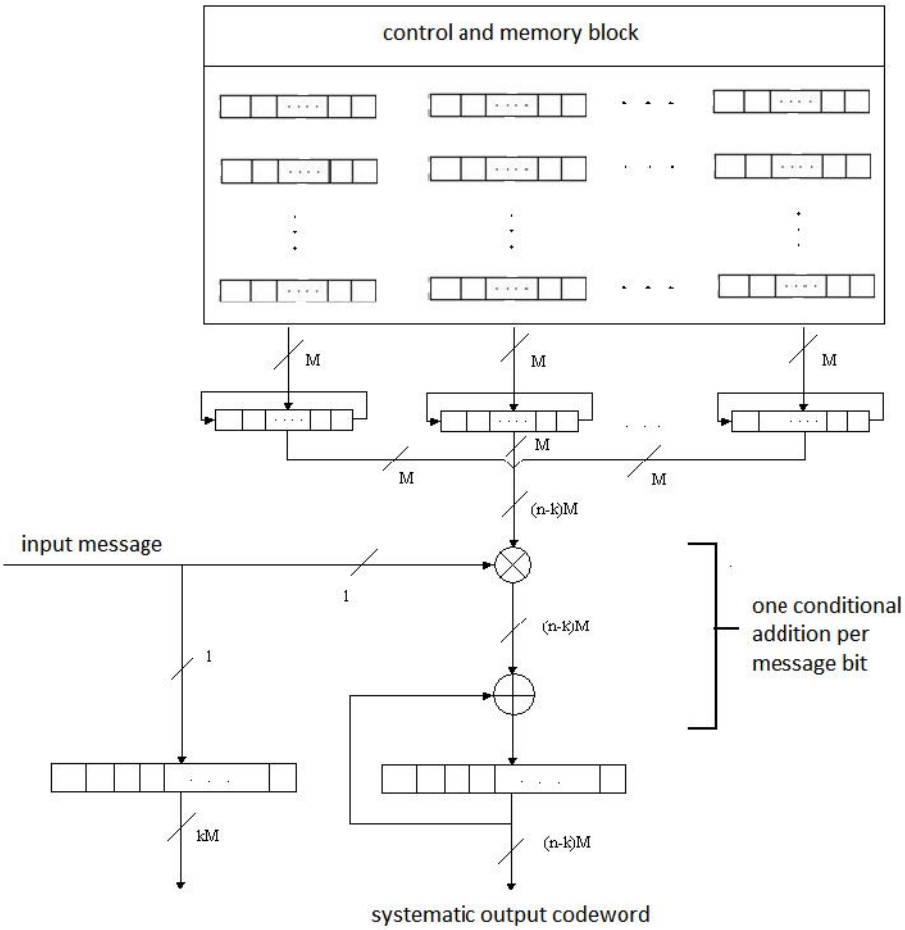


Figure 9 The hardware implementation of encoder

.

Looking to the steps and figure 9. it can be seen that the main task of encoding process can be simplified to:

1. Load the new row of generator matrix to generator register(a array of register given to accommodate the row of generator matrix needed to be multiplied with current message input)
2. Recursively XOR new row of generator matrix with the previous temporary output register data and store in the same register when the input message is 1, otherwise leave the temporary output register as it is.

This simplification process saves few intermediate register arrays and also reduces the delay in getting systematic generated output.

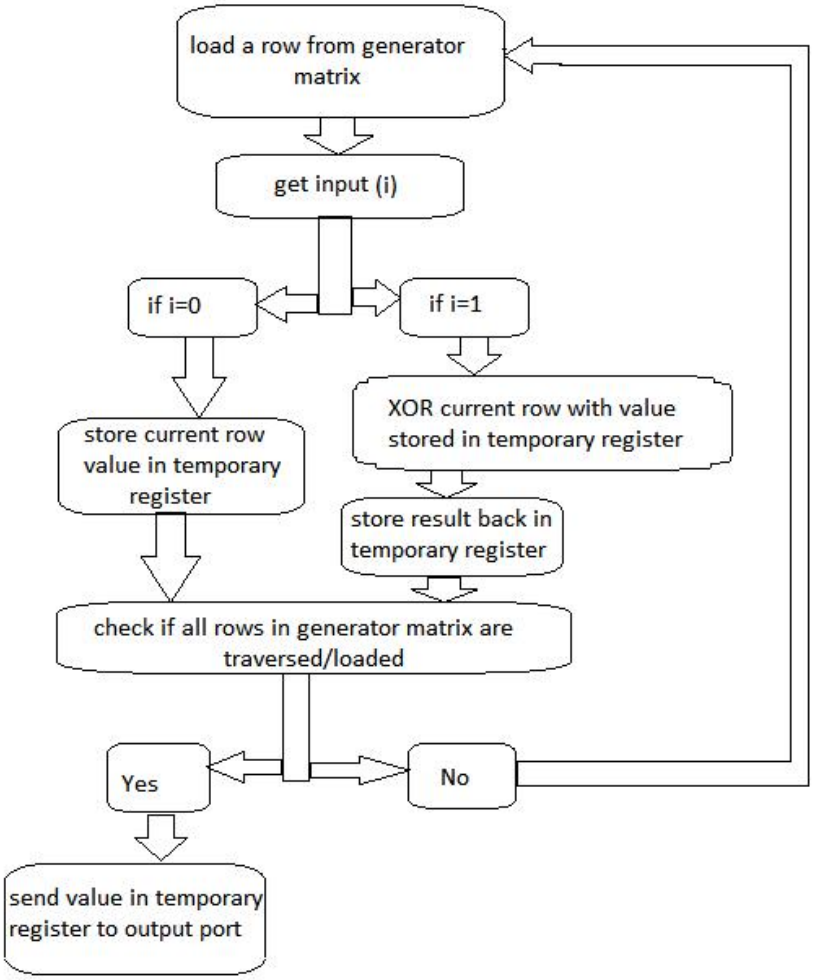


Figure 10 Flow diagram of encoder

2.5. FPGA Implementation summary

For implementation of Encoder *Xilinx XC3S500E FPGA* (Spartan 3E) board was considered. The device utilization summary is given as:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	100	5472	1%
Number of Slice Flip Flops	40	10944	0%
Number of 4 input LUTs	181	10944	1%
Number of bonded IOBs	43	240	17%
Number of GCLKs	1	32	3%

CHAPTER 3

DECODING

LDPC decoding algorithms for AWGN channels are based on Gallager's iterative decoding method. Reworking Gallager's method, MacKay came up with sum product algorithm for LDPC decoding. Belief propagation algorithm is also classified as a sum product algorithm. Sum product algorithms are presented as messages update equations on a factor graph. Factor graphs are bipartite graphs that are composed of two kinds of nodes like variable nodes for variables and factor nodes for local functions. A variable node is connected to a factor node by an edge if the variable is an argument of the local function.

3.1. Bounded Distance and Maximum Likelihood Decoding

For any linear code, A= 1, meaning that there is precisely one codeword of weight zero. For good codes, A = 0 for all j less than some value d, called the minimum distance. A code with minimum distance d can always correct $\left(\frac{d-1}{2}\right)$ errors using a bounded distance decoder (BDD). Imagine the codeword vectors as points in space. No two words are closer together than the minimum distance, d. If we draw spheres around each codeword of radius $\left(\frac{d-1}{2}\right)$, no two spheres will overlap. If no more than $\left(\frac{d-1}{2}\right)$ errors are made by the channel, the received word will lie within the sphere of the transmitted word, and thus be correctly decoded. Figure below illustrates this decoding. The smaller circles represent codewords, and the large circles a radius of $\left(\frac{d-1}{2}\right)$. The codeword in centre was transmitted.

If less than $\left(\frac{d-1}{2}\right)$ errors are made, the received word resembles the small circle labelled A and is clearly within the sphere for the desired codeword. If slightly more errors are made, the result could be something like small circles B or C. A bounded distance decoder would make an error in both of these cases. However, the circle labelled C, though outside the sphere of radius $\left(\frac{d-1}{2}\right)$ closer to the transmitted codeword than any other codeword. A maximum likelihood decoder always finds the closest codeword to the received word. Because of this, it can decode more than $\left(\frac{d-1}{2}\right)$ errors some of the time.

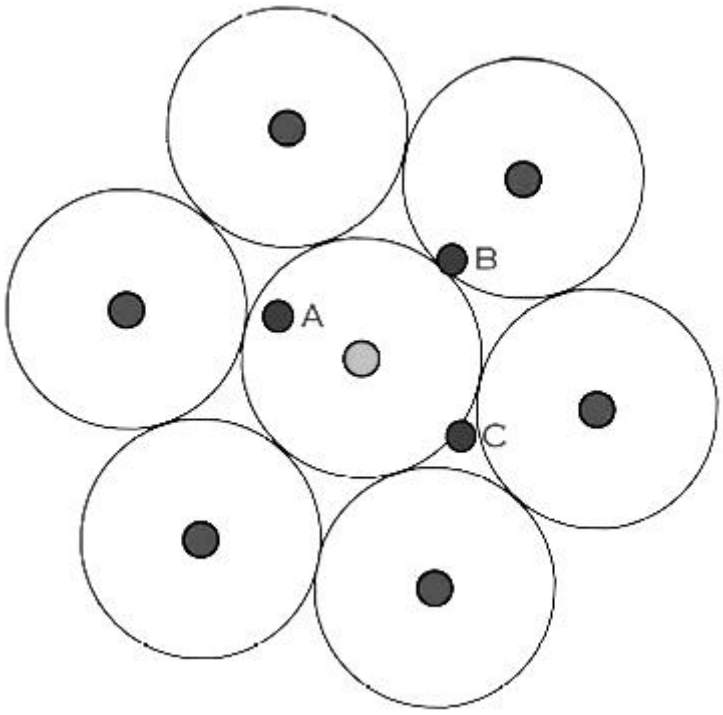


Figure 11 Message received on bounded region map

For a code of rate R and length n, there are 2^Rn codewords. A maximum likelihood decoder has to find the distance between the received word and each of the codewords in order to choose the smallest one. So, while the maximum likelihood decoder can correct the most errors, its complexity grows exponentially with the length of the codewords. For this reason, iterative decoding methods, which have complexity that still grows linearly with the length of the codewords, are much preferred.

3.2. Message Passing Decoding

Message passing is easiest to understand on the binary erasure channel (BEC). This channel introduces no errors, but erases some message bits. In the Tanner graph representation, then, each variable node either knows with certainty what its value is, or it does not. Decoding starts when the variable nodes send messages to their adjacent check nodes that indicate whether or not the variable node knows its value. The check nodes examine the messages they received from their adjacent variable nodes. If all the adjacent variables but one knew their value, the check node can determine the value of the remaining node because even parity is required. A round is completed when all the check nodes that can make this calculation send a message to the last variable node, letting it know its value. Check nodes connected to variables that all know their value can be removed from the decoding process. The cycle then repeats. With every round, more variable nodes learn their true values, until all is known or no more progress can be made. When no more progress can be made, the set of erasures remaining is known as a stopping set.

Since the deep space channel is very close to BEC characteristics, and is suitable for large size codes, the message passing decoding (sum product algorithm) was taken for implementation.

3.1. Sum product algorithm

Sum product algorithm uses the Tanner graph created from the parity check matrix H, as factor graph and sends belief messages between bit nodes (variable nodes for LDPC Tanner graph) and check nodes (factor nodes for LDPC Tanner graph). By this way, sum product algorithm determines the posterior probabilities for bit values based on a priori information, improving the accuracy of these calculations in each iteration. Check nodes and bit nodes in the Tanner graph perform computations in parallel and then communicate with each other over connections described by the edges of the Tanner graph. The messages that communication is composed of, are estimates of probabilities.

The nature of the nodes in the Tanner graph and the structure of the graphs interconnections are completely described by the number and location of ones in the parity check matrix H. The check nodes determine the probability that a parity check is satisfied if one particular data bit is set to be a one (or zero) and the other data bits have values with a probability distribution corresponding to the known a priori probabilities. The bit nodes determine the probability that a data bit has the value one

(or zero); given the information from all of the other check nodes. Only bits and checks that are related by having a one at a specific corresponding location in the parity check matrix need to be considered in these calculations .

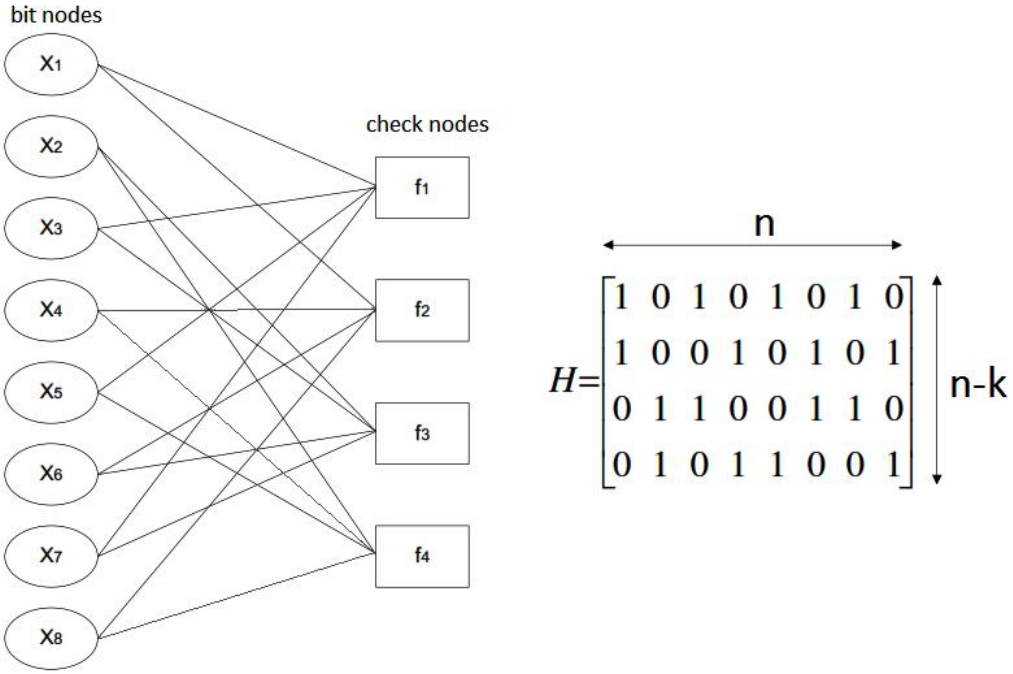


Figure 12 Messaging across the Tanner graph of parity check matrix H

R represents messages from check nodes to bit nodes and Q represents the messages from bit nodes to check nodes. Each row of parity check matrix H corresponds to a check node in the Tanner graph. In other words each row represents a single parity check of LDPC code. Similarly each column in H represents a bit node. Consequently the number of bit nodes in the Tanner graph or the number of columns in the parity check matrix is equal to the number of bits in the codeword. The location of ones and zeros in H determine the nodes which are connected in the Tanner graph.

Having a one at location row j and column i simply indicates that check node j is connected to bit node i. In the first row of H, it can be seen that there are ones in the first, fourth and seventh columns. This can be observed in the Tanner graph as connections between check node H1 (corresponding to first row in parity check matrix H) and bit nodes X1, X4 and X7(corresponding to first, fourth and seventh columns). The number of ones in a row determines the number of data inputs coming from bit nodes that the corresponding check node has. Similarly, the number of ones in a column determines the number of data inputs coming from check nodes that the corresponding bit node has.

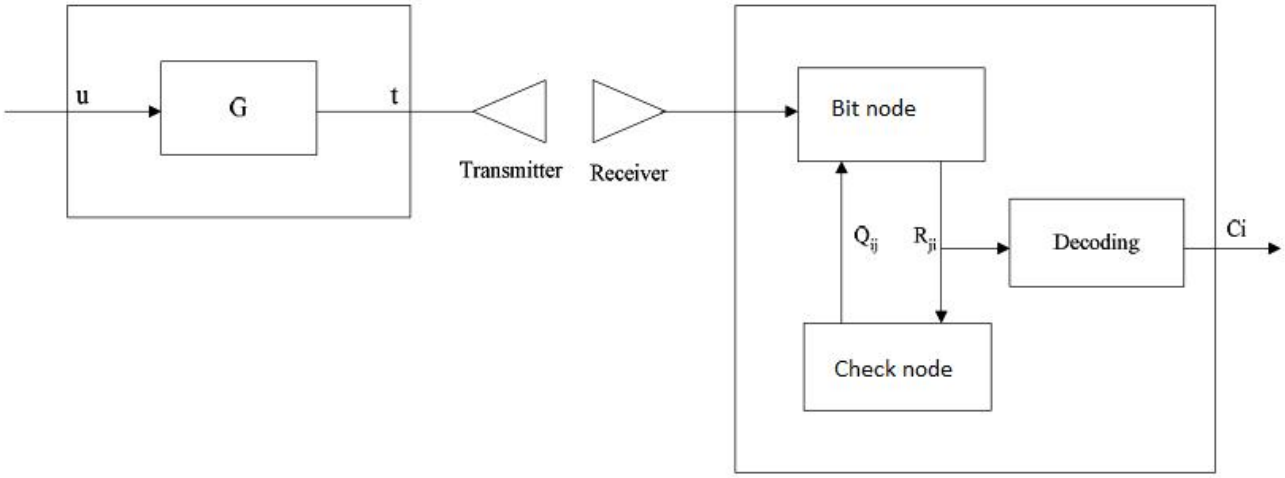


Figure 13 The general structure of LDPC encoder and iterative decoder is shown

.

As stated before, the content of messages include probability values but these probability values can be either real probability values or probability values in log domain. It is observed in the literature that sum product algorithm for LDPC decoding is classified into two main groups according to the structure of the messages between check nodes and bit nodes. These are sum product algorithm in probability domain and sum product algorithm in log domain. Details and sub groups of these main types of sum product algorithm will be described in detail in the next sections.

3.2. Sum Product Algorithm in Probability Domain

Sum Product Algorithm in Probability Domain uses real probability values in the iterative preparation of messages between check nodes and bit nodes. Algorithm works as follows:

Step 1: Messages from bit nodes to check nodes (denoted as q_{ij}) are initialized to probability values calculated according to the channel characteristics and the values of decoder input bits with AWGN. This initialization is done like equations 7 and 8 where y_i is the received data with AWGN and σ is the noise variance. $p_i^0 and\ p_i^1$ represent the apriori probabilities for each bit of the received codeword determined by the data received from the AWGN channel. For the first iteration, q_{ij} values are initialized to $p_i^0 and\ p_i^1$ values. Initialization is done once for decoding of each received codeword,

$$q_{ij}^{0} = 1 - \ p_i^1 = \ p_i^0 = \frac{1}{1 + \ e^{-2y_i/\sigma^2}}$$

… (7)

$$q_{ij}^{1} = p_i^1 = \frac{1}{1 + \ e^{2y_i/\sigma^2}}$$

… 8.

Step 2: Messages from check nodes to bit nodes are calculated. Each check node gathers all the incoming messages from bit nodes connected to it to generate r_{ij} value where r_{ji}^0 is the probability that check j is satisfied if it is assumed that data bit $t_i = 0$. Similarly where r_{ji}^1 is the probability that check j is satisfied if it is assumed that data bit $t_i = 1$. These probabilities are computed as in equations 8 and 9 The notation $i \in row[j]/\{i\}$ means the indices i' $(1 \leq i \leq n)$ of all bits in j $(1 \leq j \leq m)$ which have value one, not including the current bit index, i.

$$r_{ji}^0 = \frac{1}{2}\left[1 + \prod_{i' \in row[j]/\{i\}} (q_{ij}^0 - q_{ij}^1)\right]$$

… equation 8.

$$r_{ji}^1 = \frac{1}{2}\left[1 - \prod_{i' \in row[j]/\{i\}} (q_{ij}^0 - q_{ij}^1)\right]$$

… equation 9.

Step 3: Messages from bit nodes to check nodes are calculated. Each bit node gathers the probability information from the check nodes that are connected to it and generate the q_{ij} values, where q_{ji}^0 is the probability that data bit t_i =0, given the values of all check nodes other than j.Similarly, q_{ji}^1 is the probability that data bit t_i =1, given the values of all check nodes other than j . These probabilities are computed as shown in equations 10 and 11.

$$q_{ij}^0 = \alpha_{ij}\, p_i^0 \prod_{j' \in col[i]/\{j\}} r_{ji}^0$$

… (10).

$$q_{ij}^1 = \alpha_{ij}\, p_i^1 \prod_{j' \in col[i]/\{j\}} r_{ji}^1 \quad \ldots (11)$$

Step 4: Extrinsic probabilities of decoder output bits are calculated. These are calculated in a similar way that q_{ij} values are calculated. These extrinsic probabilities are used to determine the values for each decoder output bit. Similar to q_{ij} values calculation, the accuracy of these probabilities improves with each iteration of the algorithm.

$$Q_i^0 = \alpha_i \prod_{j \in col[i]} r_{ji}^0$$

$$Q_i^1 = \alpha_i \prod_{j \in col[i]} r_{ji}^1$$

Step 5: Decoder output bit candidates are determined according to the probability values calculated in previous step for the given condition:

$$\hat{c}_i = \begin{cases} 1 \; if \; Q_i^1 > 0.5 \\ 0 \; otherwise \end{cases}$$

Step 6: The syndrome of the decoded output candidate is calculated. As the general property of linear block codes, syndrome value indicates if the decoded output candidate is equal to the transmitted codeword. Thus, it is verified if the decoding is successful or not. The syndrome calculation is made by matrix multiplication of decoded output candidate like:

$$\hat{c}_i \times H^T = \hat{S}$$

If $\hat{S}$ is a zero vector of 1 x (N-K) then this means the received code word is decoded correctly a decoded output candidate is given out as decoder output. Otherwise, decoding continues iteratively by repeating the algorithm starting from Step 2 until the syndrome is received as zero vector. In practical

applications the number of iterations are limited to some value which is usually give as a decoder parameter called maximum number of iterations.

3.3. Sum Product Algorithm in Log Domain

Sum product algorithm in log domain is another form of sum product algorithm where the probabilities are characterized by the log-likelihood ratios (LLRs).This means, the same steps are used as sum product algorithm in probability domain in but the real probability values are replaced with LLR values. Thus, instead of r_{ij} values L(r_{ij}) values are used which are calculated as $L(r_{ij}) \triangleq \log \frac{r_{ij}^0}{r_{ij}^1}$. Similarly q_{ij} values are replaced by $L(q_{ij}) \triangleq \log \frac{q_{ij}^0}{q_{ij}^1}$, and the values of p_i and Q_{ij} are calculated in the same faishon.

The various steps of this process are described as:

Step 1: Messages from bit nodes to check nodes (denoted as L(q_{ij})) are initialized to LLR values calculated using the channel characteristics and the values of decoder input bits with AWGN. This LLR value L(p_i) is calculated like equation 12 where yi is the received data with AWGN and . is the noise variance. For the first iteration, L(q_{ij}) values are initialized to L(p_i) values calculated from a priori probability values determined by the data received from the AWGN channel.

$$L(p_i) = \log \frac{p_i^0}{p_i^1} = \frac{2}{\sigma^2} y_i$$

$$L(p_i) = L(q_{ij})$$

… (12).

Step 2: Messages from check nodes to bit nodes are calculated as LLR values. Each check node gathers all the incoming messages from bit nodes connected to it to generate L(r_{ji}) value. Before calculation of L(r_{ji}) following equations using L(q_{ji}) values following information should be given:

For independent random variables X1 and X2 the joint log-likelihood ratio $L(x_1 \square x_2)$ is given by:

$$L(x_1 \boxplus x_2) = \ln \frac{1 + e^{L(x_1)} e^{L(x_2)}}{e^{L(x_1)} + e^{L(x_2)}}$$

Consequently, the joint log-likelihood ratio $L(x_1 \boxplus \dots \boxplus x_l)$ is given as:

$$L(x_1 \boxplus \dots \boxplus x_l) = \ln \frac{1 + \sum_{i=1}^{l} \tanh(Lx_i/2)}{1 - \sum_{i=1}^{l} \tanh(Lx_i/2)} = 2.\tan^{-1}\left(\prod_{i=1}^{l} \tanh\left(\frac{L(x_i)}{2}\right)\right)$$

Thus, $L(r_{ji})$ which is composed of $L(q_{ji})$ values can be calculated like:

$$L(r_{ji}) = 2.\tan^{-1}\left(\prod_{i' \in \frac{row[j]}{\{i\}}} \tanh\left(\frac{L(q_{i'j})}{2}\right)\right)$$

… (13).

The notion $i' \in \frac{row[j]}{\{i\}}$ means the indices i'($1 \le i' \le n$) of all bits in row j ($1 \le j \le m$) which have value 1, not including the current bit index i.

Step 3: Messages from bit nodes to check nodes are calculated as LLR values. Similar to probability domain algorithm, each bit node gathers the probability information in LLR domain from the check nodes that are connected to it and generate the $L(q_{ji})$ values. These LLR values are computed as shown in equation 14. Two terms contribute to the calculation of $L(q_{ji})$ values, LLR calculated from a priori probability values used in initialization which is $L(p_i)$ and $L(r_{ji})$ values coming from check nodes.

$$L(q_{ij}) = L(p_i) + \sum_{j' \in col(i)/\{j\}} L(r_{j'i})$$

… (14).

Step 4: Extrinsic LLR values L(Q_{ji}) of decoder output bits are calculated (in a similar way used for L(q_{ji})) for determining decoder output bits. Similar to L(q_{ji}), the accuracy of these values improves with every iteration.

$$L(Q_i) = L(p_i) + \sum_{j' \in col(i)/\{j\}} L(r_{j'i})$$

Step 5: Decoder output bit candidates are determined according to the probability values calculated in previous step for the given condition:

$$\hat{c}_i = \begin{cases} 1 \; if \; Q_i^1 > 0.5 \\ 0 \; otherwise \end{cases}$$

Step 6: The syndrome of the decoded output candidate is calculated. As the general property of linear block codes, syndrome value indicates if the decoded output candidate is equal to the transmitted codeword. Thus, it is verified if the decoding is successful or not. The syndrome calculation is made by matrix multiplication of decoded output candidate like:

$$\hat{c}_i \times H^T = \hat{S}$$

If $\hat{S}$ is a zero vector of 1 x (N-K) then this means the received code word is decoded correctly a decoded output candidate is given out as decoder output. Otherwise, decoding continues iteratively by repeating the algorithm starting from Step 2 until the syndrome is received as zero vector. In practical applications the number of iterations are limited to some value which is usually give as a decoder parameter called maximum number of iterations.

The state machine diagram for this algorithm can be summarised as:

Initialization:- input from channel are loaded into the bit processor blocks
Bit to Check:- bit node processor performs Equation 13
Check to Bit:- check node processor performs Equation 12
Output:- After a number of iterations or satisfied syndrome check, output message is generated.

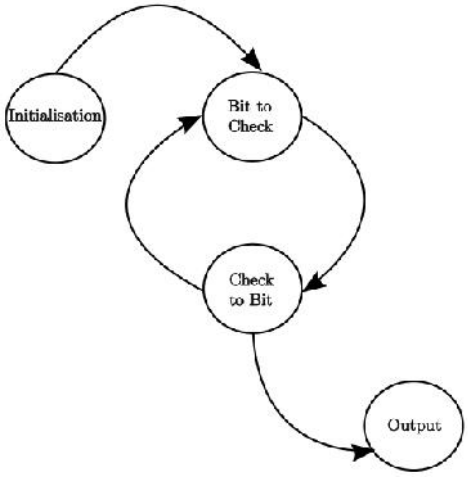


Figure 14 State machine diagram of sum product algorithm

3.4. Hardware implementation of decoder

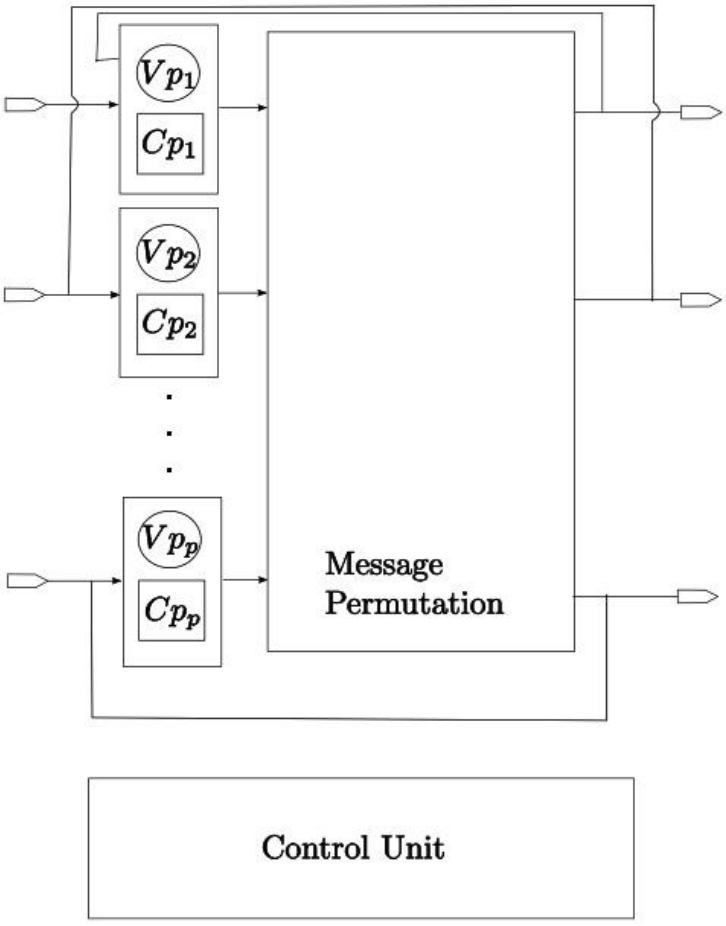


Figure 15 Top block diagram of decoder

The architecture consists of a number (P) of processors, a message permutation block and a control logic block, as seen in figure 15. There is a smaller number of bit (check) processors then bit (check) nodes in the Tanner Graph meaning that each bit (check) processor is assigned a subset of these nodes. The processors themselves are responsible for storing the incoming messages, performing the node operations and forwarding the outgoing messages, while the assignment of the nodes to processors is handled by the control unit.

The decoding process follows four distinct parts as shown in state machine figure 14. The bit to check and check to bit half iterations are repeated a predetermined number of times before outputting the decoded codeword. The number of iterations is likely to be small, around ten to keep the decoding time small. With a small number of iterations, the benefit of early termination is likely to be outweighed by the increase in cycle time.

3.5. Look up table approximation method:

The function given as Y(x) is implemented using this method.

$$Y(x) = \log(1 + e^{-|x|})$$

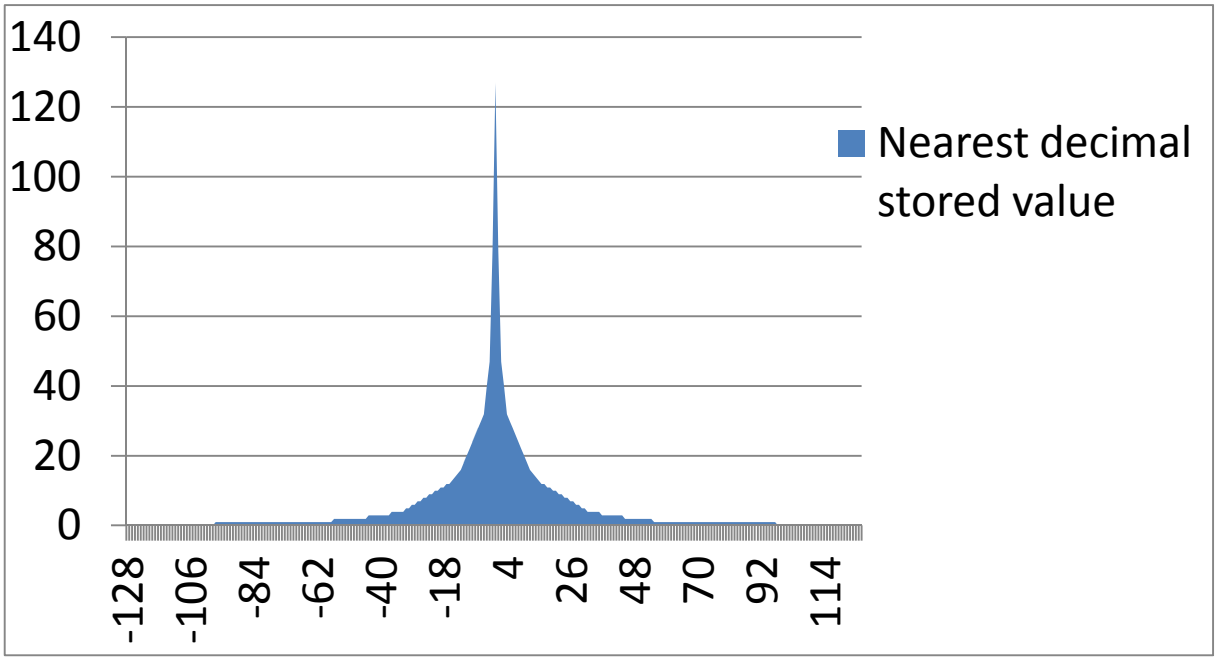


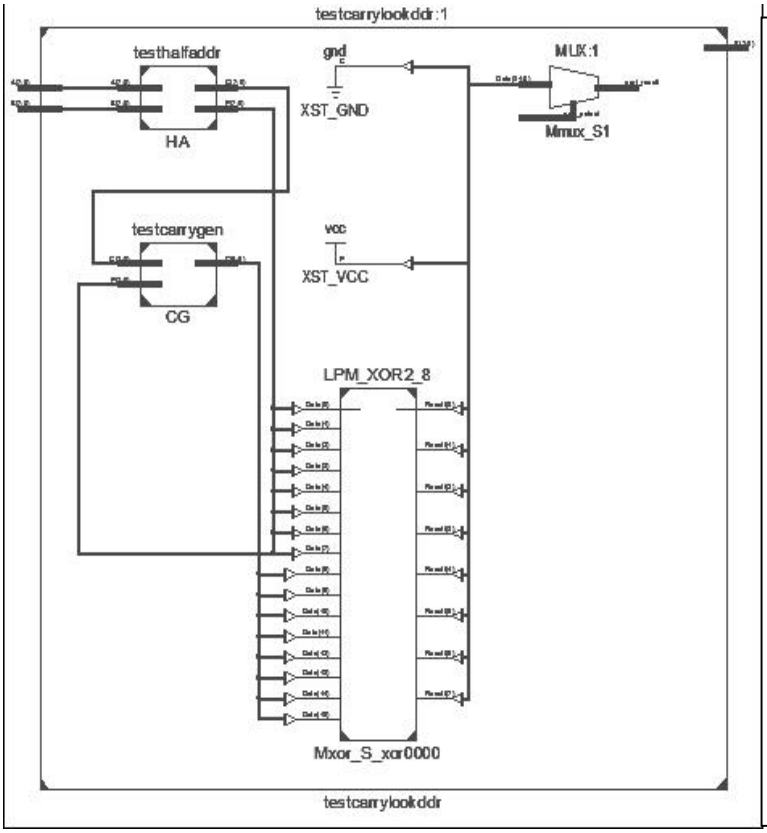
Figure 16 Look up table approximationfor given function

.

The values taken for input extends from -8 to +7.9375, making the increments of 0.0625. This allows the total number of message input levels extends upto 256, ranging -128 to +127. The other look up tables are generated in the similar manner.

3.6. Adders

For the LDPC decoder, there is a need to add multiple input messages in parallel. The number of inputs to the adder dictates the maximum supported degree weight of the bit and check nodes. The maximum degree weight for the check nodes is the number of inputs into the adder, while for the the bit nodes it is one less, due to one of the inputs being used for the incoming channel measurement.



For the inputs A and B
G = Ai * Bi (Carry Generation)
P = Ai + Bi (Carry Propagation)
Ci+1 = Gi + (Pi * Ci)

In VHDL
Testhalfaddr:
P <= A xor B
G <= A and B
Testcarrygen:
For i= 0 to 7
tempC(i+1):= G(i) or (P(i) and tempC(i))
Then
C <= tempC

Figure 17 Carry look ahead adder architecture

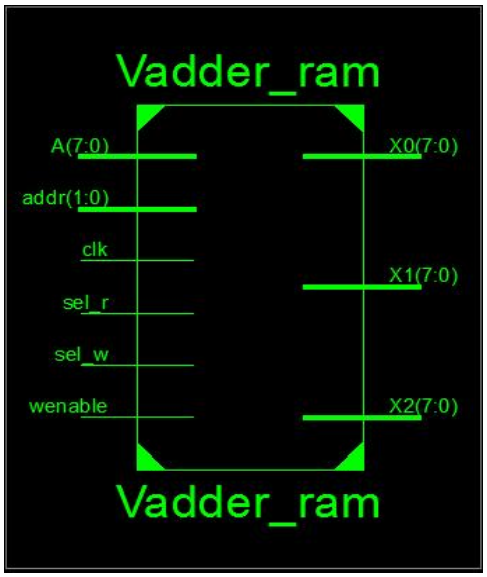
For the prototype it was decided to handle the parallel inputs with tree adders, as opposed to carry save adders, due to the complexity of the latter and the small number of operands. Ripple, carry

look ahead and carry select adders were investigated and ranked on their performance and complexity. It was found that the carry select adder was the fastest and most complex (and consumed the most power), while ripple adder was the slowest but least complex. The carry look ahead adder was the best compromise between speed and complexity and so it was chosen for the decoder.

The carry look ahead adder calculates the carry signals in advance, based on the input signals. The implementation of the carry look ahead adder is based on the VHDL code provided as:

.

3.7. RAM and memory




```
if  wenable = 1  then
if  sel_w = 0 then
Memory0 (addr ) <=  A
else
Memory1 (addr ) <=  A
if (sel_r='1') then
X0 <= memory1(0)
X1 <= memory1(1)
X2 <= memory1(2)
else
X0 <= memory0(0)
X1 <= memory0(1)
X2 <= memory0(2)
```

Figure 18 Bit node adder RAM unit

.

The different RAM units used in the decoding process are:

1. Adder RAM unit: In order to process one message per clock cycle, the adder must have all messages associated with a bit node available. In order to achieve this the Adder RAM unit implements a serial to parallel converter, making the messages associated with the current node being processed available. To avoid stalling the processor for every bit node, the converter has two memories. It serially loads the messages for the next bit node into one memory while the other memory with the messages for the current bit node is available in parallel for the adder.

2. Codeword RAM unit: There are two codeword RAM's, with the control line selecting which one is available to the bit node adder and which is available to load code-bits. With this configuration the decoder is able to load the next codeword while it is still processing the current one.

3.8. Bit node processor and control:

The bit node processor is responsible for receiving messages from the check node through the message permutation block, processing the messages and outputting them to the message permutation block. It is important to note that the bit node processor is responsible for keeping track of the individual bit nodes in the code. From the perspective of the LDPC control unit, the bit nodes send a stream of messages with no distinction as to which belong to what bit node.

The bit node processor has a control signal that controls its function, reading messages into the message RAM or sending messages. The signal only has effect when the processor is coming out of reset. Figure 19 shows the bit node processor unit top level RTL schematic.

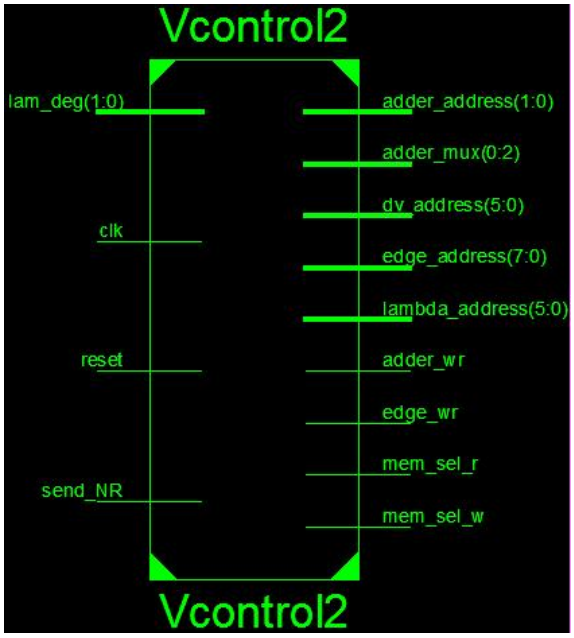


Figure 19 Bit node processor unit top level RTL schematic

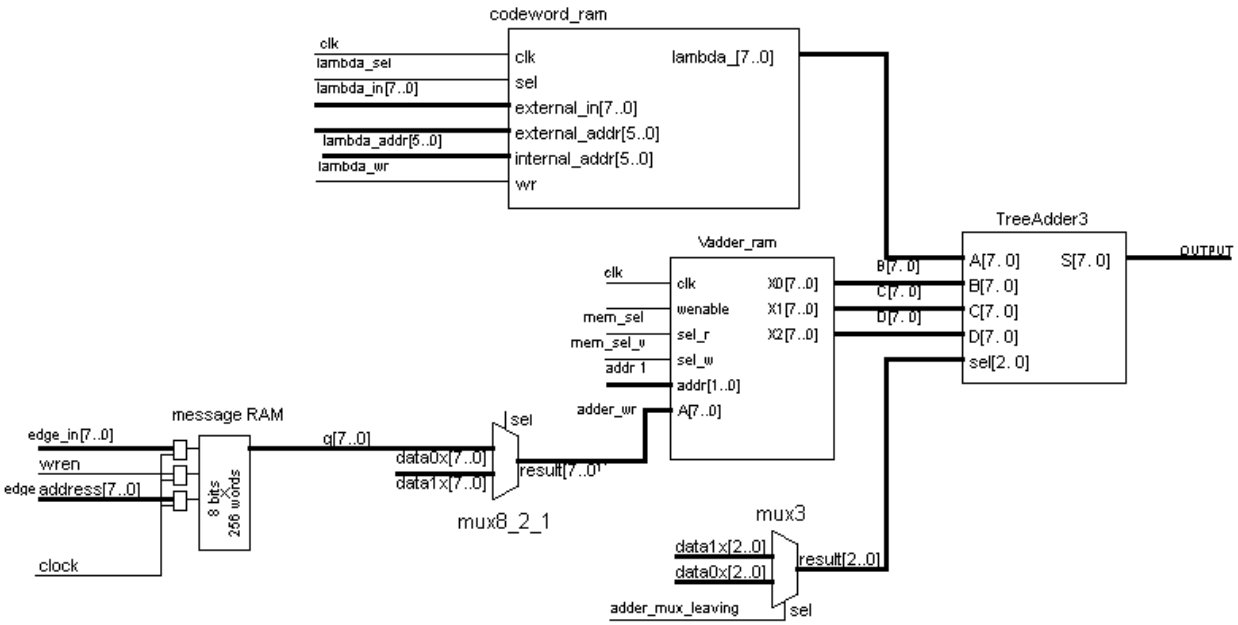


Figure 20 Bit node control unit

3.8. Check node processor and control

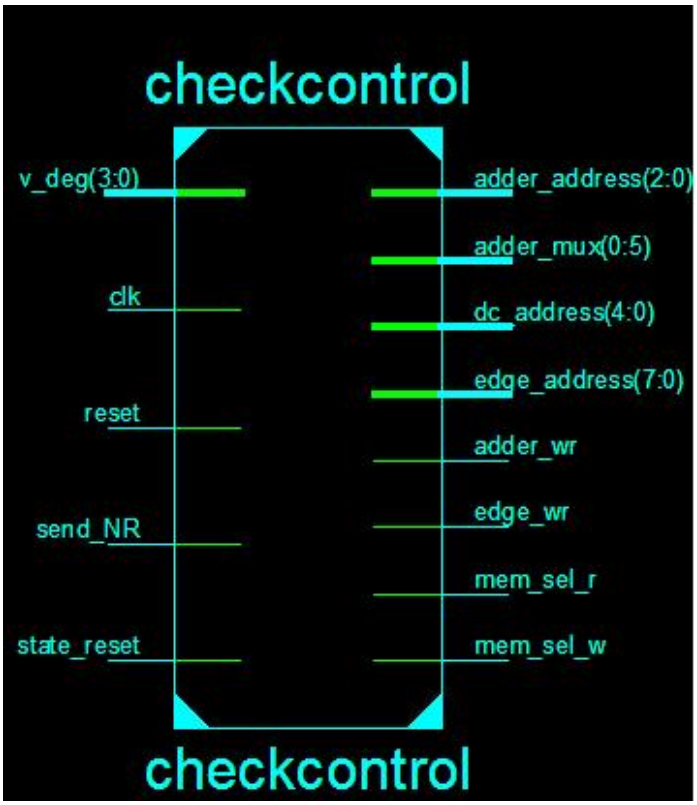


Figure 21 Top level RTL schematic of the check node processor

The check node processor is identical to the bit node processor except for the adder and there being no codeword RAM or its associated control unit. For the check node processor the adder accepts 8 inputs (only 6 are used in the prototype). At each adder input has a Ã(x) lookup table, the output of the adder is passed through another Ã(x) lookup table which also performs the sign correction as described in Section 3.7. The sign correction is performed by XOR'ing the sign bit (MSB) of the inputs together and if the result is a `1', then the result of the LUT is made negative.

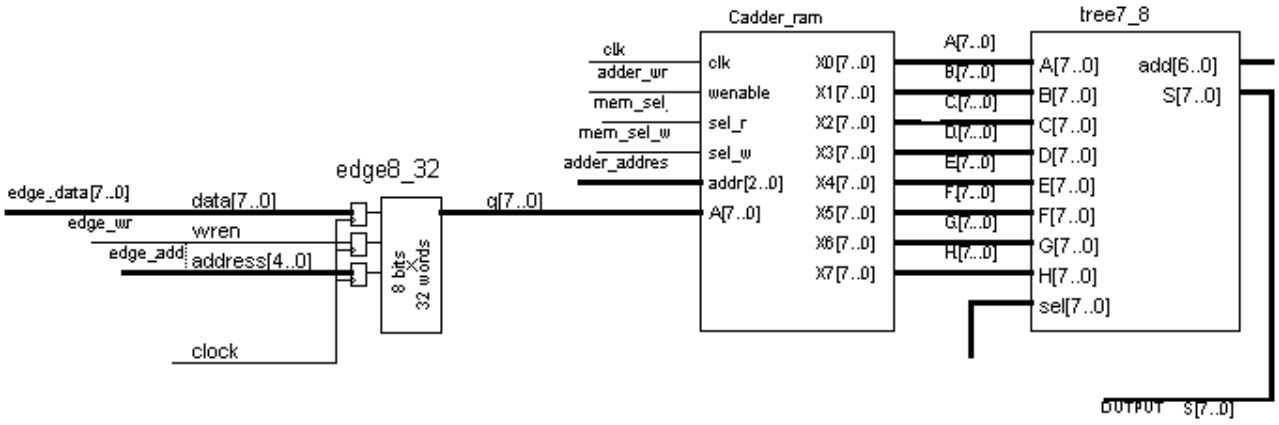


Figure 22 Check node processor control unit

.

When the check node processor is identical to the bit node processor except for the adder and there being no codeword RAM or its associated control unit. For the check node processor the adder accepts 8 inputs (only 6 are used in the prototype). At each adder input has a Ã(x) lookup table, the output of the adder is passed through another Ã(x) lookup table which also performs the sign correction as described in Section 3.7. The sign correction is performed by XOR'ing the sign bit (MSB) of the inputs together and if the result is a `1', then the result of the LUT is made negative. Figure 21 shows the block diagram of the check node processor.

3.9. Control unit

The control unit performs the following tasks:

1. When the bit node processor is receiving messages, the control unit sets the write enable for the message RAM and increments the address, ensuring the incoming messages are stored in the correct location.

2. When the processor is processing and sending messages, the control unit loads the address for the next bit node into the adder RAM unit.
3. The control unit increments the address of the channel measurements RAM block so that the channel measurement associated with the currently processing bit node is available to the adder.
4. The control unit increments the address for the degree weight RAM block, which is used by the control unit to determine how many edges to load into the Adder RAM unit for each bit node.
5. When the messages of the bit node are being calculated, the control unit disables the corresponding adder input, eliminating the effect of incoming message from the outgoing message.
6. On the first iteration the message RAM is uninitialized, so the main control unit asserts a control signal which bypasses the message RAM via a multiplexer.

The main control unit implements a 12 stage state machine, controlling the all of the parts of the decoder.

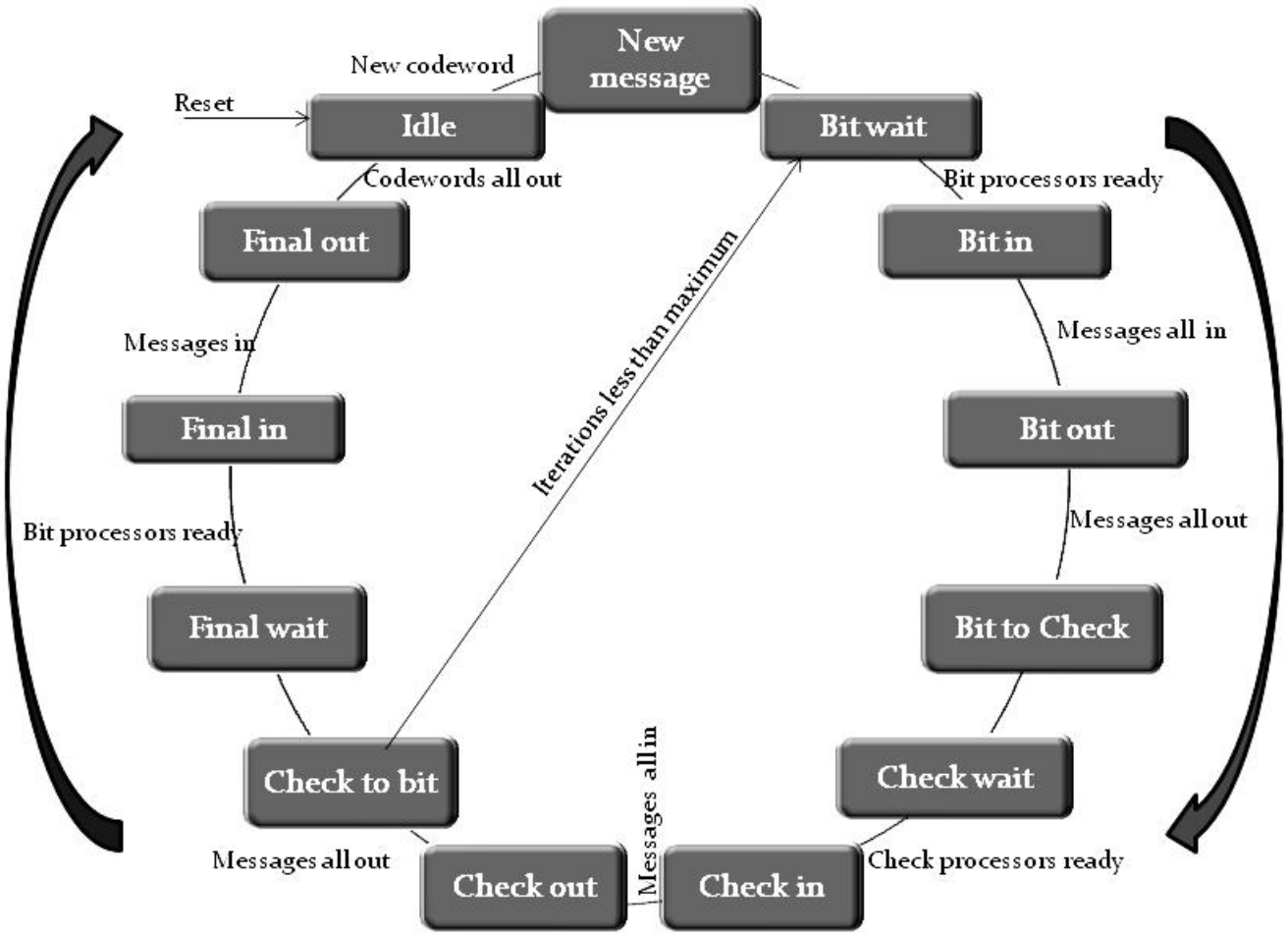


Figure 23 Main control unit state machine

.

Idle

The decoder starts in this state, and remains here until a new codeword is available to decode. In this state writing to the permutation network is disabled. When a codeword is available the decoder proceeds to the new message state, flipping the codeword RAM select signal so that the new message is available to the bit node processor. The bit and check node processors are held in the reset state.

New message

In this state the control unit sets a signal to bypass the bit node message RAM blocks as they are uninitialized. The decoder proceeds straight into the bit wait state and drops the reset on the bit node processors.

Bit wait

With the reset dropped on the bit node processors they start sending messages, but there is a latency introduced of one bit node (in the prototype this is 3 clock cycles) by the bit node adders so the decoder must wait until the bit node processors output messages.

Bit in

In this state write is enabled into the permutation block and the messages from the bit node processor are written in. The control block increments the address of the switch ROM so that the the bit node messages are stored in the correct interleaver blocks.

Bit out

The reset on the check node processors is dropped while the writing to the message permutation block is disabled. The decoder increments the address of the switch and interleaver ROMs so that the check node can receive the correct message.

Bit to check

In this state the functions of the bit node and check node processors are reversed. The check node will be sending messages and the bit node receiving. The input into the message permutation block is set to the check node processors. The check node processors are reset for a clock cycle to switch them from receiving to sending messages.

Check wait

As in the bit wait state, the decoder has to wait one check node (in the prototype this is 6 clock cycles) for the check node processors to start sending messages.

Check in

In this state write is enabled for the permutation message block and the messages from the check node processors are stored in the interleaver blocks. The control block increments the address of the switch ROM to ensure that the check node messages are stored in the correct interleaver blocks.

Check out

The bit node procssors' reset is dropped while writing to the message permutation block is disabled. The decoder increments the address of the switch and interleaver ROMs so that the bit node processors can receive the correct messages. When all of the messages have been loaded into the bit node processors' message RAM the decoder proceeds to the check to bit state.

Check to bit

In this state the functions of the bit node and check node processors are reversed. The bit node processors will be sending messages and the check node processors receiving. The input into the message permutation block is set to the bit node processors. The bit node processors are reset for a clock cycle to switch them from receiving to sending messages. If the iteration count is less then the

predetermined number (10 in the prototype) the decoder moves to the bit wait state, otherwise it moves to the final wait state.

Final wait

This state is similar to the bit wait state, the reset being dropped on the bit node processors, the decoder is waiting until the bit node processors start sending messages. The control unit sets a signal that causes all of the inputs into the adder of the bit node processor to be used. In doing this the bit node processors calculate the messages ready for hard decision decoding.

Final in

In this state the messages from the bit check nodes are stored in the interleaver banks. The input switch ROM is unused however, with the input from each processor being stored in its respective interleaver bank. When all the messages have been sent to the interlever banks the decoder proceeds to the final out state.

Final out

The decoder increments the address of the output switch and interleaver ROMs. This produces the decoded codeword on the 1st output of the permutation network. The decoder output block performs a hard decision on the codeword and stores the result in the decoded RAM. The decoder has now decoded a codeword and proceeds back to the beginning, the idle state, to process another.

CHAPTER 4

SIMULATION RESULTS AND ANALYSIS

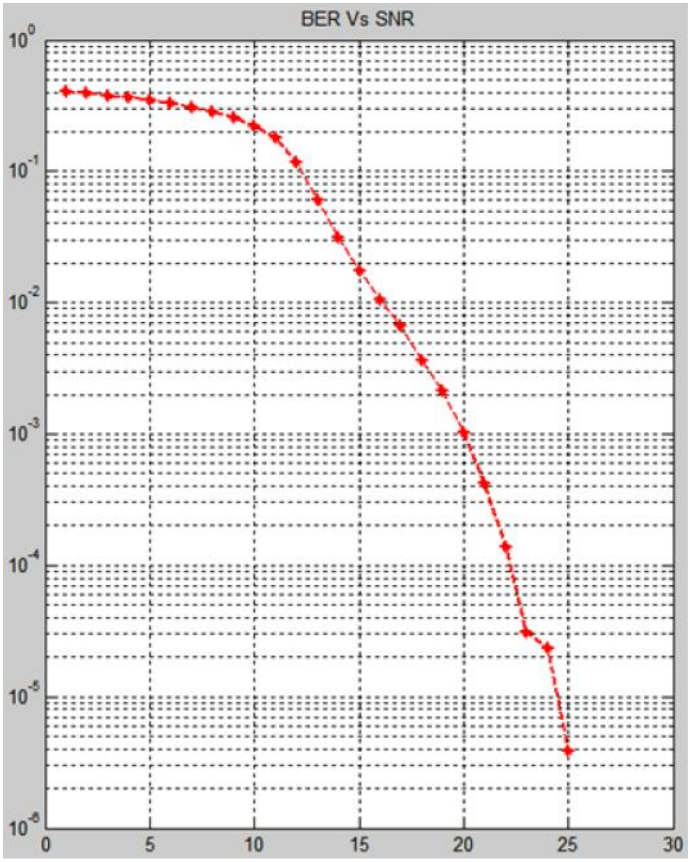


Figure 24 BER plot for rate ½ matrix with block size of 128 bits for AR4JA code

BER plot for AR4JA

BER plot for modified AR4JA

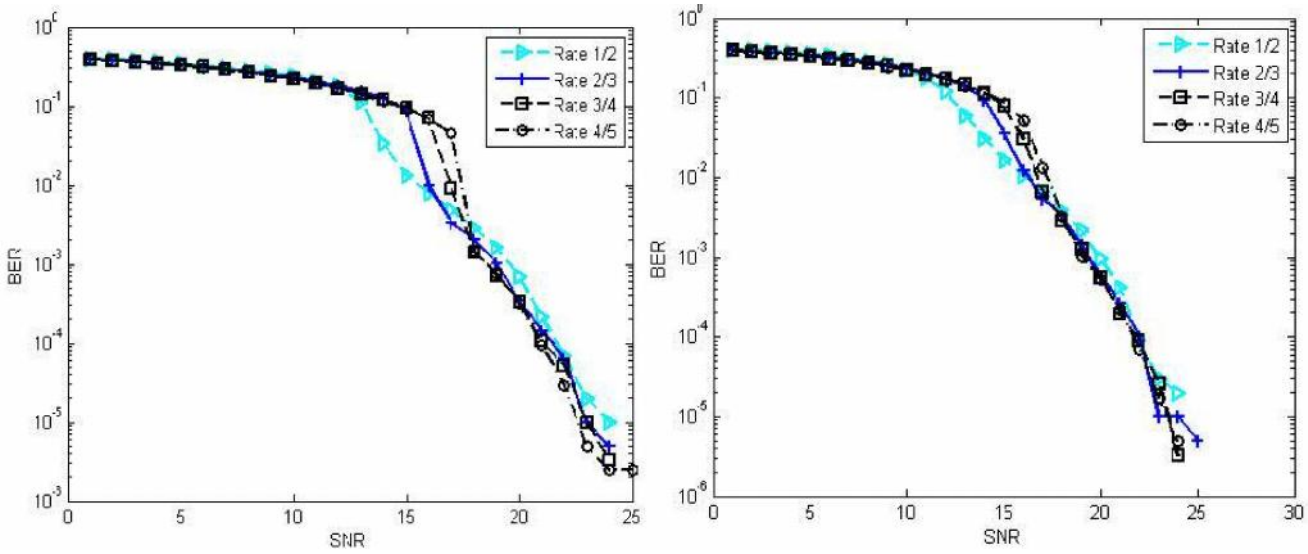
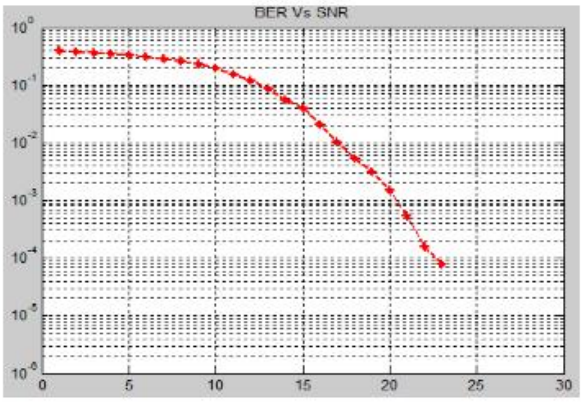
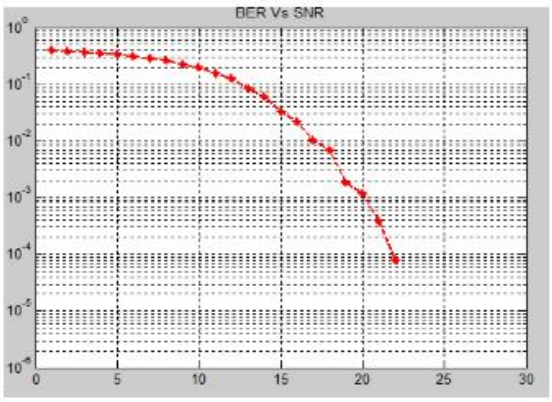


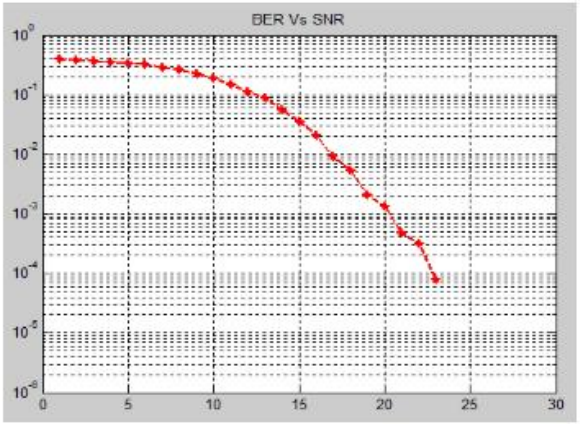
Figure 25 Comparison between BER plots for different rates of matrix with block size of 128 bits



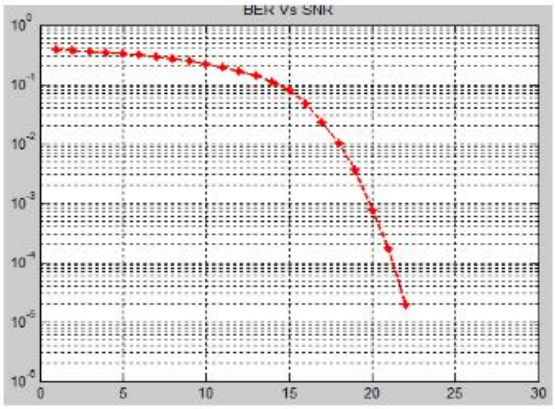
(No. Of Iteration 10)



(No. Of Iteration 20)



(No. Of Iteration 50)



(No. Of Iteration 100)

Figure 26 BER plot at different values of iteration for code configuration : AR4JA, Block size- 128, code rate ½

Analyzing the results for code 3(AR4JA) and code 4(modified AR4JA)

Table 3 Output code block length and encoding time for different rates of AR4JA

Code Rates	N	AR4JA Encoding time
1/2	2560	0.132261
2/3	3584	0.157087
3/4	4608	0.175156
4/5	5632	0.206130

Table 4 Output code block length and encoding time for different rates of modified AR4JA

Code Rates	N	Modified AR4JA Encoding time
1/2	2560	0.122104
2/3	3584	0.150088
3/4	4608	0.162018
4/5	5632	0.185178

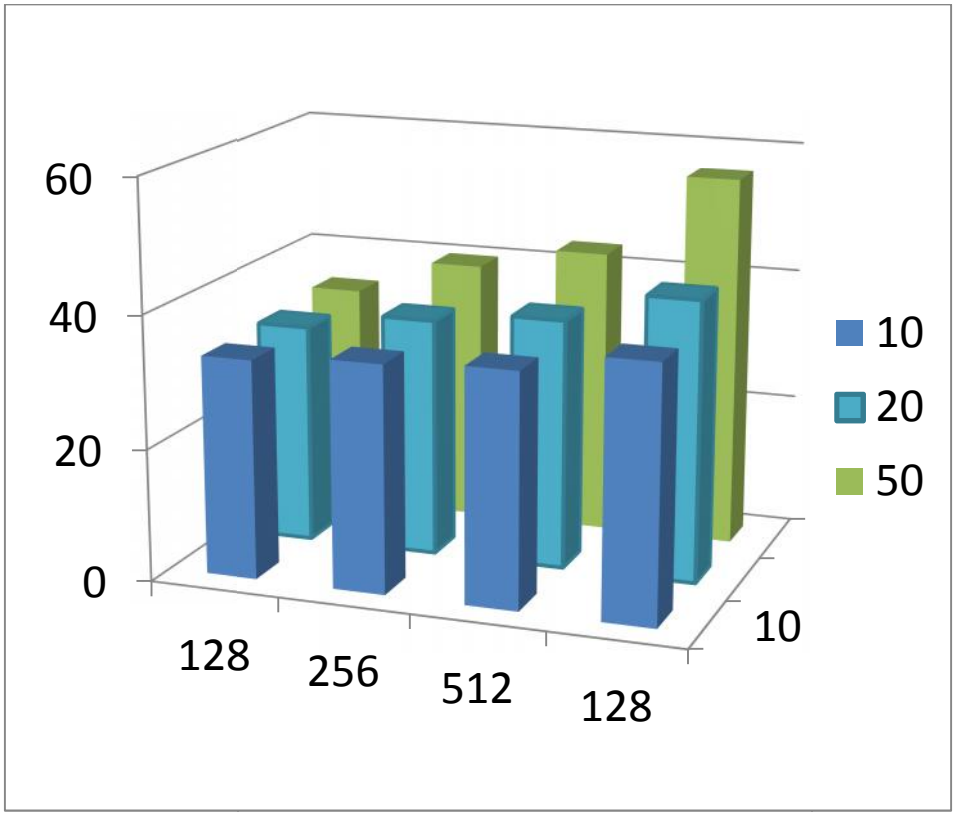


Figure 27 Variation between bit size of a circulant (rate 1/2) and decoding time for different number of iterations

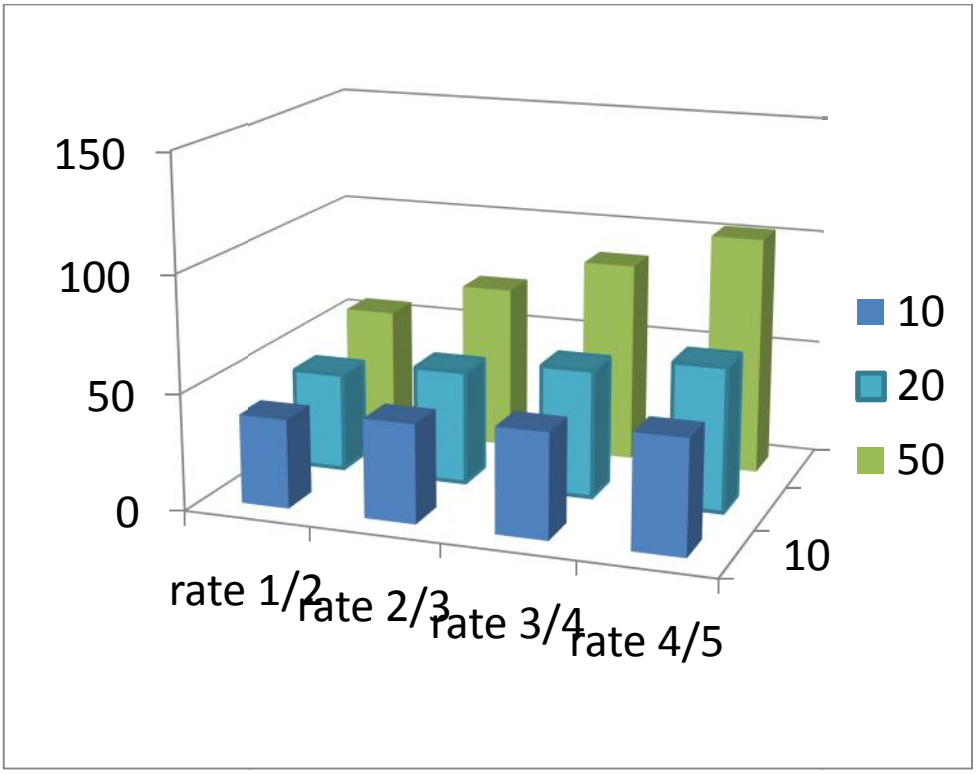


Figure 28 Variation between different rates of a circulant (size 128 bit) and decoding time for different number of iterations

CHAPTER 5

CONCLUSIONS

1. For lower SNR values the BER fall is higher than in higher SNR values.
2. For the different rates of a block size, the fall in BER values happens earlier and is more steep than that of higher rates.
3. Increasing the total number of iterations the fall in BER values gets more and more steep indicating better and more error free decoding.
4. On increasing number of iterations for a given code configurations, the decoding time increases .
5. Increasing block size and\or rate increases the decoding time.
6. The BER plot for the MATLAB and VHDL are almost the same(neglecting the small number of samples taken)
7. It can be seen from decoder structure that high block size results in more number of flip flops / memory cells, the smaller block size is preferred.

CHAPTER 6

FUTURE WORK AND SCOPE

The next step in the series would be to get the BER plot of decoded message to be very close to Shannon's limit. This task can be achieved by further optimization of decoder architecture. For this reason a new decoding algorithm Belief Propagation can also be used.

Then the next focus could be on implementation for larger number of bits *i.e.* larger H matrix and hence more number of message bits are processed at a time, resulting in faster system.

Further development would be to use this system in more sophisticated and advanced communication systems like modulation schemes such as OFDM, DVB-S2 and 802.3 and(Wi-Max) or the devices like blue-ray discs or DTH television systems.

There are developments been going to use this code family in many deep space communication systems varying across different rates and block sizes.

CHAPTER 7

REFERENCES

[1] R. Gallagar “Low Densit y Parit y Check Code” *IRE Transaction paper Theory* pp 21-28, Jan 1962.

[2] R.M.Tanner “A recursive approach to Low complexit y codes” *IEEE Trans. Information Theory*,

pp. 533-547 Sept 1981.

[3] Consultative Committee for Space Data Systems (CCSDS). Low density Parity Check Codes for use in Near-Earth and Deep Space Applications, Sept. 2007. Orange book, available at

http://public.ccsds.org/publications/archive/131x1o2e1.pdf

[4] Z. Li. “Efficient Encoding of Quasi-Cyclic Low-Density Parity-Check Codes.” *IEEE Transactions on Communications 54, no. 1 (January 2006): 71-81.*

[5] S. Lin and D. Costello, Jr. *Error Control Coding. 2nd ed. New Jersey: Pearson* Prentice Hall, 2004.

[6]D. Divsalar, S. Dolinar, and C. Jones. “Construction of Protograph LDPC Codes with Linear Minimum Distance.” In *Proceedings of the IEEE International Symposium on Information Theory (Seattle, Washington). Piscataway, NJ: IEEE,* July 2006.

[7] M. Mansour and N. Shanbhag, \High-throughput LDPC decoders," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 11, pp. 976{996, Dec. 2003

[8]S. Johnson, “Introduction to ldpc codes," in ACoRN Spring School on Coding, Multiple User Communications and Random Matrix Theory, 2006

[9]K. Zhang, X. Huang and Z. Wang, “High-throughput layered decoder implementation for quasi-cyclic LDPC codes,” *IEEE J. Selected Areas Commun., vol. 27, no. 6, Aug. 2009*

[10]Z. Wang, Z. Cui, “Low-complexity high-speed decoder design for quasicyclic LDPC Codes,” *IEEE Trans. VLSI Systems, vol. 15, no. 1, Jan. 2007*

[11]Abhishek Kumar,Madhusmita Mishra,Sarat Kumar Patra” Performance Evaluation and Complexity analysis of Re-jagged AR4JA code over AWGN channel” International Journal of Scientific & Engineering Research, Volume 4, Issue 6, June 2013 ISSN 2229-5518

[12]K. Andrews, S. Dolinar, and J. Thorpe. "Encoders for Block-Circulant LDPC Codes." In *Proceedings of the IEEE International Symposium on Information Theory (Adelaide, Australia)*, 2300-2304. Piscataway, NJ: IEEE, September 2005.

[13]J. Lee and J. Thorpe. "Memory-Efficient Decoding of LDPC Codes." In Proceedings of the IEEE International Symposium on Information Theory (Adelaide, Australia), 459-463. Piscataway, NJ: IEEE, September 2005.

[14]N. Wiberg, "Codes and Decoding on General Graphs", Ph.D. dissertation, Linkoping University, Sweden, 1996.

[15]W. E. Ryan, "An introduction to LDPC codes", Department of Electrical and Computer engineering, University of Arizona, Aug. 2003

[16]S.Papaharalabos ,P.Sweeney,B.G.Evans, *"Modified sum product algorithm for decoding LDPC"* ,IET Comm ,2007

[17]Hu Y. and E.Eleftheriu, "Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC codes", Global Telecommunications Conference 2001,Volume 2, pp. 1036, 2001

[18]Karkooti M. and J.R. Cavallaro, "Semi-parallel reconfigurable architectures for real- time LDPC decoding", IEEE International Conference on Information Technology 2004, Volume 1, pp. 579 – 585, April 2004.

[19]Lee W.L. and A.Wu, "VLSI implementation for low density parity check decoder", IEEE International Conference on Electronics, Circuits and Systems 2001, Volume 3, pp. 1223-1226, 2001.

[20]Theocharides T., G.Link and E.Swankoski, "Evaluating alternative implementations for LDPC decoder check node function", IEEE Computer society Annual Symposium on VLSI 2004, pp.77-82, 2004.

[21]Wymeersch H., H. Steendam and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)", IEEE International Conference on Communications 2004, Volume 2, pp. 772-776, June 2004.

[22]Karkooti M. and P. Radosavljevic, "Configurable, High Throughput, Irregular LDPC Decoder Architecture: Tradeoff Analysis and Implementation", International Conference on Application-specific Systems, Architectures and Processors 2006, pp. 360-367, September 2006.

[23]Liao E.,Y. Engling, B. Nikolic,"Low-density parity-check code constructions for hardware Implementation" , IEEE International Conference on Communications, Volume 5, pp. 2573-2577, June 2004.